LANDesk® Management Suite 8.6 Understanding Batch File Distribution Packages

Revision 1.4

Jared Barneck
November 16, 2006

Information in this document is provided in connection with LANDesk Software products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in terms and conditions for such products, LANDesk Software assumes no liability whatsoever, and LANDesk Software disclaims any express or implied warranty, relating to sale and/or use of LANDesk Software products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. LANDesk Software products are not intended for use in medical, life saving, or life sustaining applications. The reader is advised that third parties can have intellectual property rights that can be relevant to this document and the technologies discussed herein, and is advised to seek the advice of competent legal counsel, without obligation of LANDesk Software.

LANDesk Software retains the right to make changes to this document or related product specifications and descriptions, at any time, without notice. LANDesk Software makes no warranty for the use of this document and assumes no responsibility for any errors that can appear in the document nor does it make a commitment to update the information contained herein.

Copyright © 2006, LANDesk Software, Ltd. All rights reserved.

LANDesk, Targeted Multicast, and Peer Download are registered trademarks or trademarks of LANDesk Software, Ltd. or its controlled subsidiaries in the United States and/or other countries.

*Other brands and names may be claimed as the property of others.

Table of Contents

Introduction

Batch files are powerful scripts that can automate almost any task an administrator needs to perform on any number of systems. Batch files resemble early programming languages, including support for IF statements, FOR loops, GOTO statements, variables, and more.

The amount of software tools available to batch files has increased dramatically since the days of DOS. Recent Windows® versions give batch files access to thousands of tools created by Microsoft® and thousands more are made available by third party software vendors. With all of these tools available to batch files, an infinite amount of tasks can be accomplished.

Scope

This document discusses the following batch file related topics:

- Batch File Overview
- Understanding the Microsoft® Local System Account
- Understanding the ERRORLEVEL Variable
- Sending Status to the Core Server
- Handling Batch File Reboots
- Passing Inventory Data as a Parameter
- Troubleshooting
- Sample Batch Files

Assumptions

This document is written to explain how batch files are handled by LANDesk® as a Distribution Package. This document is not designed to teach batch file development. It is designed to help batch file developers understand how to integrate a batch file with LANDesk and learn the LANDesk features that have been created to enhance this integration.

It is assumed that the reader is already familiar with writing intermediate to advanced batch files. Some reader may need to study batch file concepts in order to understand this document.

It is assumed that the reader has already read the Users Guide or the help file, especially the Software Distribution section and is familiar with configuring Distribution Packages.

An anonymous access web share and/or a UNC share for storing Distribution Package files should already be set up.

Batch File Overview

A batch file is a text file ending with a .bat or .cmd extension. The contents of a batch file can be any command or list of commands that run from a command prompt. A batch file can be a simple one-line command or complex string of commands including any executable. The batch file may also call variables, IF statements, FOR loops, and other internal commands.

Most batch files begin with @ECHO ON or @ECHO OFF. By design LANDesk silences all batch files so @ECHO ON or @ECHO OFF does not at first seem to make a difference. However, when a batch file is pushed with LANDesk, it logs everything that is passed to STDOUT to a log file on the client. For this purpose it is a good idea start batch files with @ECHO ON. (For more information, see the section on Troubleshooting later in this document.)

Batch files can and should have comments. It is good practice to use comments (lines beginning with REM or ::) to document the batch file and its use.

Batch file scripts execute the commands in order for the most part, starting at the top and going down; however, IF statements, FOR loops, GOTO statements, labels, and parameters can change this order, while REM or :: can cause a line to be ignored.

Batch Files

The following are examples of two batch files. One is called a basic batch file and the other a complex batch file. In this manual the term "basic batch file" means a batch file that runs only one line and the term "complex batch file" means a batch file that runs more than one line.

Example 1 – The Basic Batch File

The following basic batch file runs only one command line. Lines are numbered for discussion purposes.

```
1. @ECHO ON
2. :: Batch file 1 is written to copy a file
3. copy c:\file1 c:\file2
```

This is batch file is the simplest of batch files, running only one task. This batch file could be written with only one line, but following "good practice" guidelines, ECHO is set and a comment describing the batch file's function is included.

Example 2 – A Complex Batch File

Though this batch file really is not that complex, it demonstrates the ability to add complexity to a batch file by using IF statements, labels, and GOTO commands. Again, lines are numbered for discussion purposes only.

```
1.   @ECHO ON
2.   :: This batch file checks for the existence of file1
3.   :: and if it finds it, it copies file1 to file2
4.   IF EXIST c:\file1 (
5.         copy c:\file1 c:\file2
```

```
6.        IF "%ERRORLEVEL%"="0" (GOTO success) ELSE (GOTO fail)
7.     ) ELSE (
8.        GOTO fail
9.     )
10.  :fail
11.  ECHO The file "file1" does not exist
12.  GOTO end
13.  :success
14.  ECHO The file "file1" was successfully copied
15.  :end
```

This batch file checks for the existence of file1 in line 4. If file1 exists, lines 5 and 6 run. Line 6 checks whether the copy command succeeded and if so sends processing to the "success" label on line 14, at which point line 15 runs, echoing a custom success message before the batch file ends.

If file1 exists but line 5 fails, line 6 sends processing to the "fail" label on line 11. Line 12 runs, which echoes a custom error message. Line 13 sends processing to the "end" label, skipping the "success" section, and the batch file ends.

If file1 does not exist, line 8 runs which sends processing to the "fail" label on line 11. Line 12 runs, which echoes a custom error message. Line 13 then sends processing to the "end" label, skipping the "success" section, and the batch file ends.

This document is not designed to train an administrator on how to write a batch file. This document expects the readers to already know how to write batch files, or to learn more about writing batch files of their own accord.

To learn more about writing batch file scripts, see Microsoft's TechNet site on Administration and Scripting Tools and Using Batch Files. Many other third party websites exist that provide both batch file training and samples.

Deployment Method Differences

When deploying batch files in a scheduled task, the account under which the batch runs depends on the delivery method. It is important to understand the account under which the batch file runs for development and troubleshooting.

Depending on the Delivery Method and the user permissions, a batch file can run as either Microsoft's Local System account or as the Logged on user.

To understand the differences between running a batch file as Microsoft's Local System account and as the Logged on user see the following section entitled Understanding the Microsoft Local System Account.

Deploying Using a Push Delivery Method

When a batch file is deployed using a push Delivery Method, the batch file runs as the account under which the client machine's LANDesk Management Agent service runs. By default, this is Microsoft's Local System account.

Deploying Using a Policy

When a batch file is deployed using a policy Delivery Method, the LANDesk agent software determines if the logged on user is an administrator on the client machine and if so, the batch file runs as the logged on user. If the user is not an administrator, the batch file runs as Microsoft's Local System account.

Deploying Using a Policy-Supported Push

A policy supported push is combination of a push and a policy. A batch file deployed with a Policy-Supported Push starts out using the push delivery method. When a batch file is deployed using a push Delivery Method, the batch file will run as the account under which the client's LANDesk Management Agent service runs. By default, this is Microsoft's Local System account.

The scheduled task becomes a policy only for machines that do not complete the initial push. On machines that the batch file runs as a policy, the LANDesk agent software determines if the logged on user is an administrator on the client machine and if so, the batch file will run as the logged on user. If the user is not an administrator, the batch file will run as Microsoft's Local System account.

Understanding the Microsoft® Local System Account

When using LANDesk to push a batch file a LANDesk administrator should understand the account under which the batch file runs. When pushed with LANDesk, or when using a policy and users are not local administrators, this account is Microsoft's Local System account.

Microsoft recommends that services running on client computers in a domain run as the Local System account. The Local System account is not a user account and running batch files as Local System is quite different than running a batch files as a user.

When a Software Distribution task is pushed using LANDesk, the Core Server contacts the LANDesk services on the client computers. These services are running under the Local System account and this account is used to run the batch file.

For more information on Microsoft's Local System account, see the following two MSDN® web sites:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/localsystem_account.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ad/ad/the_localsystem_account.asp

Testing Batch Files Using Microsoft®'s Local System Account

Batch files should be tested as Local System and not as the logged in user when they are either pushed with LANDesk Management Suite, or deployed using a policy to users that are not local administrators.

To test a batch file as Local System, a command prompt can be opened that is running as Local System. LANDesk has written KB Article 2614 that demonstrates how to open a command prompt as a Local System. These steps are repeated in greater detail here.

Opening a Local System Command Prompt with Localsch.exe

To open a Local System command prompt with localsch.exe, logon to a client computer as a user that is a member of the local administrators group.

Go to Start > Run. Enter cmd and click OK. This opens a command prompt running as the logged on user.

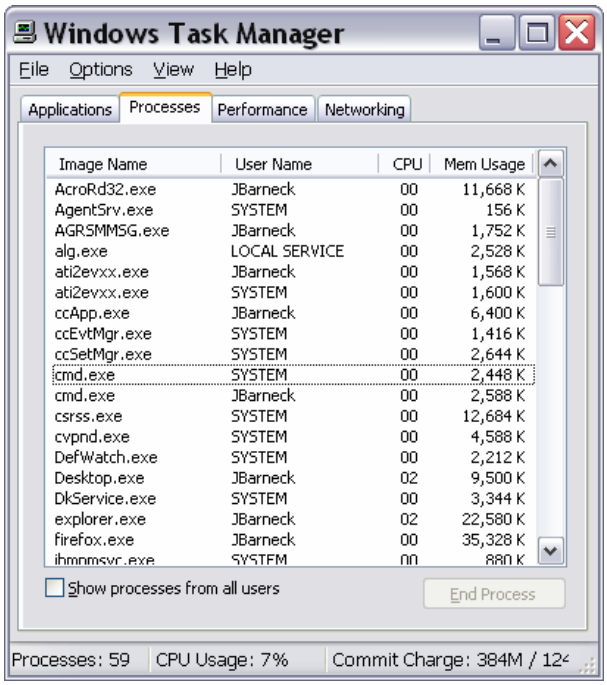
Run the following command to launch a command prompt running as Local System:

```
"C:\program files\LANDesk\LDClient\localsch.exe" /exe=cmd.exe
```

This does NOT launch the command prompt immediately but launches it within the next few minutes. Please be patient. When the time arrives, cmd.exe runs, opening a command prompt running as Local System.

To verify that the command prompt is running as Local System, open Task Manager and click on the Processes tab. Click on the Image Name heading to sort by name. Two cmd.exe processes are running. One command prompt is launched from Start > Run and the User Name field shows the process is running as the logged on user. The other command prompt is configured to launch using the Local Scheduler Service which runs as Local System, so the cmd.exe process runs as Local System as well.

Under the User Name field it should say SYSTEM.



A batch file should be tested from this Local System command prompt to verify that it will run successfully when pushed with LANDesk.

Opening a Local System Command Prompt with Microsoft®'s AT.exe

Microsoft's at.exe command can also be used to schedule another cmd.exe task to start any time in the future.

To open a Local System command prompt with at.exe, logon to a client computer as a user that is a member of the local administrators group.

Go to Start > Run, enter cmd, and click OK. This opens a command prompt running as the user.

Use at.exe to schedule a second cmd.exe to launch one minute from the current time. To do this, use the following syntax:

```
at hh:mm /interactive cmd
```

Replace hh with the appropriate hour, 1-24, and mm with the appropriate minute, 0-60, representing a time one minute in the future.

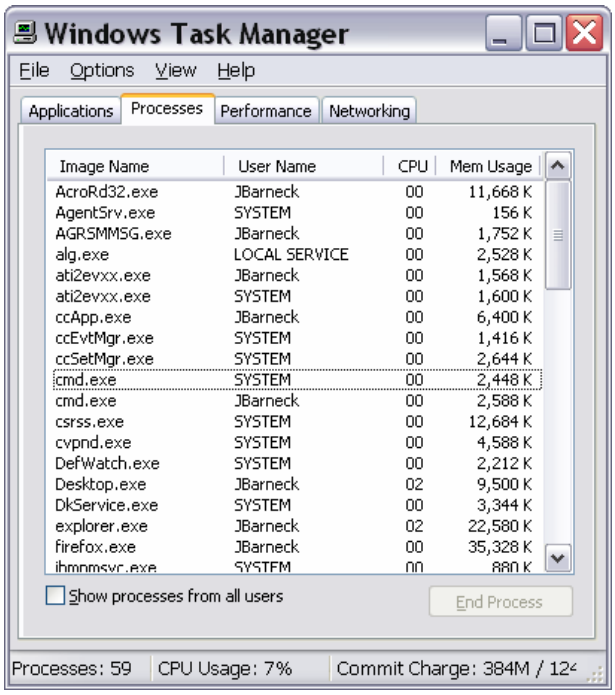
Example of the command if the current time is 1:24 pm.

```
at 13:25 /interactive cmd
```

This does NOT launch the command prompt immediately but launches it when the time specified arrives. Please be patient.

When the time arrives, cmd.exe runs, launching a command prompt that runs as Local System.

To verify that the command prompt is running as Local System, open Task Manager and click on the Processes tab. Click on the Image Name heading to sort by name. Two cmd.exe processes are running. One command prompt is launched from Start > Run and the User Name field shows the process is running as the logged on user. The other command prompt is launched by the Intel Local Scheduler service which runs as Local System, so the cmd.exe process runs as Local System as well. Under the User Name field it should say SYSTEM.



The batch file should be tested from this Local System command prompt to verify that it will run successfully.

Testing Share Permissions as Local System

When a batch file running as Local System accesses a UNC share in a domain, it accesses that share as the computer account in the domain, which in Active directory® is a member of the Domain Computers group. Providing the Local System account access to a share in Active Directory® can be done by adding the Domain Computers group to both the share permissions and the NTFS permissions. Domains without Active Directory® do not include the Domain Computers group, and networks without a domain do not have this group either. In these environments a null session share can be created (refer to LANDesk KB Article 2025). When testing permissions, both share and NTFS permission can restrict access to a file or directory. Local System needs access to both the share and any folder or files accessed.

To demonstrate this, view the example below:

```
1. @ECHO ON
2. copy \\server\share\folder1\install.msi c:\temp\install.msi
3. IF NOT “ERRORLEVEL”==“0” GOTO :end
4. copy \\server\share\folder1\data1.cab c:\temp\data1.cab
5. IF NOT “ERRORLEVEL”==“0” GOTO :end
6. c:\temp\install.msi
7. :end
```

This batch file accesses two files on a share. To demonstrate how to verify that the Local System account has access to the share, the folder, and the files used in the above batch file, follow the steps below:

1. Open a Local System command prompt using the steps given previously.

   Run the net use command to map a drive to the share.

   ```
   C:\> net use * \\server\share
   ```

   Note: When running net use with an asterisk (*) instead of a drive letter, the next available drive letter is used.

2. Change to the mapped drive. For this example, assume that drive is G.

   ```
   C:\> G:
   ```

3. Now change directories to folder1.

   ```
   G:\> CD folder1
   ```

4. Now try to copy the files to a local drive.

   ```
   G:\folder1> copy install.msi c:\temp\install.msi
   G:\folder1> copy data1.cab c:\temp\data1.cab
   ```

If all of these steps work, then Local System account has access to this share, folder, and file.

If at any point the result is either of the following errors…

- Access denied
- Network path not found

…then Local System does not have access to this path. It is the administrator’s responsibility to modify permissions necessary to give the Local System account access.

NOTE: Be aware the second error can be caused by either lack of proper permissions or using an invalid path. Verify the path is correct before assuming a lack of proper permissions

Using Environment Variables under the Local System Account

Environment variables can be used in batch files. There are many environment variables that are created by default in Windows®. Not all of the environment variables are available as Local System. Only system environment variables are available when running as Local System. Local System does not have user variables.

To demonstrate this, open a command prompt running as Local System and run SET to display the environment variables.

```
C:\WINDOWS\system32> set

ALLUSERSPROFILE=C:\Documents and Settings\All Users
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=LD-USER1
ComSpec=C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_CHECK=NO
LDMS_LOCAL_DIR=C:\Program Files\LANDesk\LDClient\Data
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
Path=C:\Program Files\Windows Resource
   Kits\Tools\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;
   C:\Program Files\WinSCP3\
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 15 Model 2 Stepping 7, GenuineIntel
PROCESSOR_LEVEL=15
PROCESSOR_REVISION=0207
ProgramFiles=C:\Program Files
PROMPT=$P$G
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\WINDOWS\TEMP
TMP=C:\WINDOWS\TEMP
USERPROFILE=C:\Documents and Settings\NetworkService
windir=C:\WINDOWS
```

Open a command prompt running as the logged on user and run SET. Compare the output of the SET command as Local System versus the logged on user.

```
C:\Documents and Settings\User1>set
ALLUSERSPROFILE=C:\Documents and Settings\All Users
APPDATA=C:\Documents and Settings\User1\Application Data
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=LD-USER1
ComSpec=C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_CHECK=NO
HOMEDRIVE=C:
HOMEPATH=\Documents and Settings\User1
LDMS_LOCAL_DIR=C:\Program Files\LANDesk\LDClient\Data
LOGONSERVER=\\LDUT2KDC03
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
Path=C:\Program Files\Windows Resource
   Kits\Tools\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Pro
   gram Files\Executive Software\Diskeeper\;C:\Program
```

```
   Files\Rational\common;C:\Program Files\Rational\ClearQuest;C:\Program
   Files\WinSCP3\
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 15 Model 2 Stepping 7, GenuineIntel
PROCESSOR_LEVEL=15
PROCESSOR_REVISION=0207
ProgramFiles=C:\Program Files
PROMPT=$P$G
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\DOCUME~1\User1\LOCALS~1\Temp
TMP=C:\DOCUME~1\User1\LOCALS~1\Temp
USERDNSDOMAIN=LANDESK.COM
USERDOMAIN=LANDESK
USERNAME=User1
USERPROFILE=C:\Documents and Settings\User1
windir=C:\WINDOWS
```

The differences between the Local System environment variables and the logged on user environment variables are marked in red. Notice that there are more environment variables available to a logged on user than are available to Local System. These variables cannot be accessed by Local System and should not be used when writing a batch that is pushed with LANDesk.

Some variables, such as %TEMP% exist for both Local System and the logged on user but the values are different. The value assigned to the variable under the Local System account is the value that is used when pushing the batch file with LANDesk.

NOTE: The client's LDClient directory, C:\Program Files\LANDesk\LDClient by default, is temporarily added to the PATH when pushing a batch file as a software distribution task. (See the next section)

Understanding the Directories in PATH

PATH is an environment variable that contains a list of directories that are searched when an executable is called. If an executable is not found in the current working directory, the directories listed in the PATH environment variable are searched. If an executable is found in any of these directories, it is launched. This is important because files called from a directory in PATH do not need to be called using the full path.

When pushing a batch file with LANDesk, the C:\Program Files\LANDesk\LDClient directory is added to PATH. This allows for LANDesk client executables such as ldiscn32.exe, sdclient.exe, localsch.exe, vulscan.exe, and others to be called without having to use the full path.

Understanding the Current Working Directory

By default the current working directory is the directory from which the batch file is called. When pushing a batch file with LANDesk, the batch file is transferred to client's sdmcache directory. The folder structure of the share or web share is

maintained in the sdmcache directory. For example, a batch file located on a web share as HTTP://CoreServer/share/batch.bat or a UNC share such as \\CoreServer\Share\batch.bat downloads to the client and the full path to the batch file is as follows:

C:\Program Files\LANDesk\LDClient\sdmcache\share\batch.bat

When the batch file is launched on the client, the current working directory is the directory the batch file is in:

C:\Program Files\LANDesk\LDClient\sdmcache\share\

This is the current working directory when the batch file begins. If CD or CHDIR is called the current working directory changes. The following batch file demonstrates this:

```
1.  @ECHO ON
2.  :: This batch file demonstrates the current working directory
3.  ECHO %CD%
4.  ::Echoes C:\Program Files\LANDesk\LDClient\sdmcache\Webshare\
5.  CD c:\temp
6.  ECHO %CD%
7.  ::Echoes C:\temp
```

This is important because files called from the current working directory do not need to be proceeded by the full path. When files are called that are not in the current working directory or not in PATH, they must be called using the file's full path.

Getting the Path to the LDClient Directory

Often the client is installed to an alternate directory instead of the default directory. The default directory is C:\program files\LANDesk\LDClient. It is important when making calls to the client directory that a variable is used to get this path, otherwise, batch files will fail on computers that have the client installed to an alternate path.

LANDesk includes an LDMS_LOCAL_DIR environment variable that maps to the LDClient\data directory. This variable contains the correct path to the LDClient directory. This path can be parsed from this variable with the following syntax:

```
%LDMS_LOCAL_DIR:~0,-5%
```

NOTE: For more information on parsing a variable in this manner, run SET /? from a Windows command prompt.

Since the LDClient directory is automatically added to the PATH variable, it is not necessary to enter the full path when running executables contained in LDClient from a batch file. However, the LDClient directory may need to be called for other reasons, such as when adding local scheduler tasks using localsch.exe. This variable can be used to call any file in the LDClient directory.

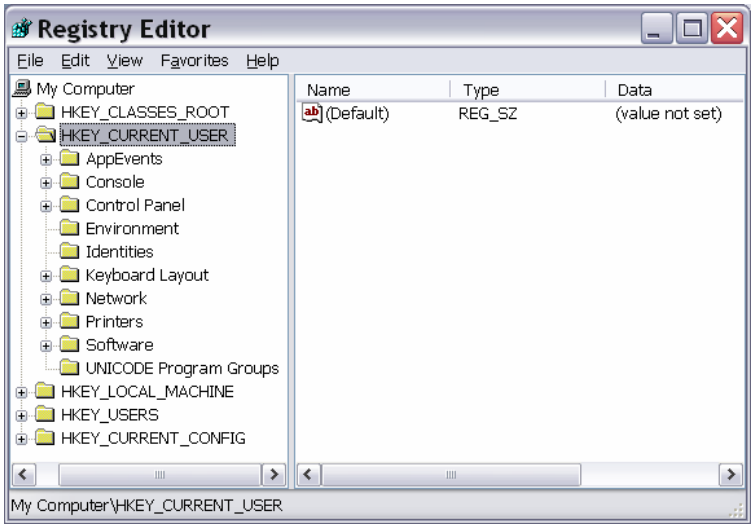
The following is an example of a Local Scheduler task that uses this parsed variable. This task runs amclient.exe from the ldclient directory to check for policies.

```
LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\amclient.exe" /taskid=1001
/cmd="/apm /s /ro" /freq=28800
```

Accessing the Registry under the Local System Account

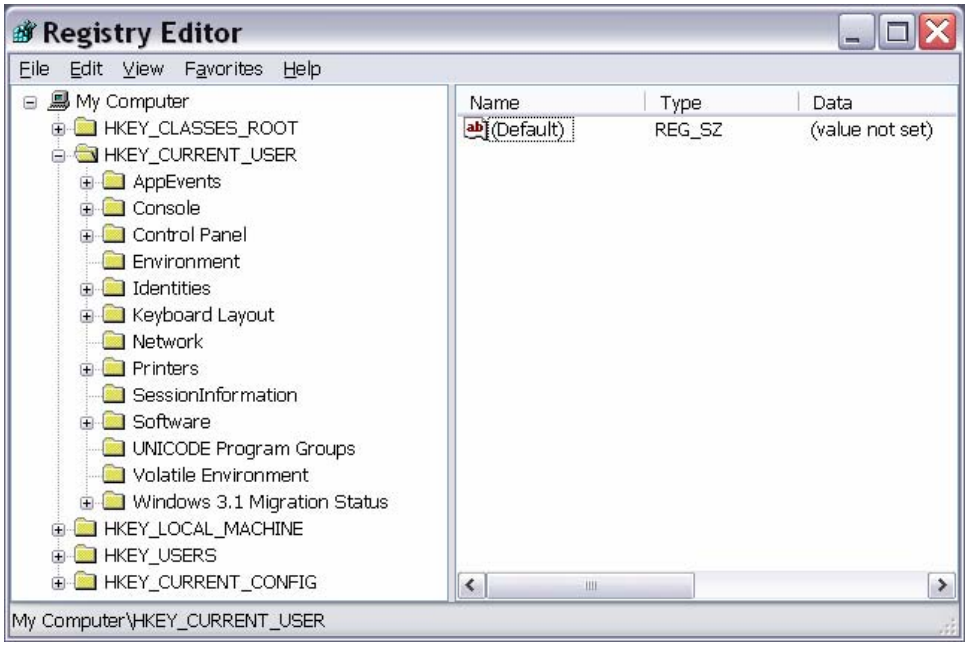
By default, every entry in the registry can be accessed and modified with Windows® commands such as regedit.exe, regedt32.exe, reg.exe or any other executable or installer that modifies the registry. However, it is important to understand that Local System loads the default users HKEY_CURRENT_USER (HKCU) hive.

To demonstrate this, launch regedit.exe from the Local System command prompt. Expand the HKCU registry key and look at the available keys:



The Local System account loaded the Default User's HKCU hive. It only contains ten subkeys.

Close regedit and reopen it from the logged on user's command prompt.



Notice that there are thirteen subkeys under the logged on user's HKCU hive. It is important to remember that when modifying registry keys in a batch file, any modifications made to HKCU are made to the Default Users HKCU hive and not the logged on user's HKCU hive. To demonstrate this, create a file named test.reg containing the following text:

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\test]
"test"=dword:00000001
```

Try importing test.reg using the regedit.exe under the Local System command prompt.

```
C:\> regedit /s c:\test.reg
```

The command is successful but the registry key is not entered into HKCU for the logged on user. The key is added to the HKCU hive for the Default User. To verify that this key is added, look under HKEY_USERS\.DEFAULT and verify that the "test" key exists.

Run the same command under the logged on user's command prompt. Assuming permission and policies allow registry edits, the key is added to the logged on user's HKCU hive.

If modifications are made to a key that exists for the logged on user but not for the Default User, this modification may fail. To demonstrate this, use reg.exe (available by default in XP) to query for a key as Local System.

```
C:\> reg query HKCU\SessionInformation
```

This fails with the following error:

```
Error:  The system was unable to find the specified registry key or value
```

This failure occurs because the HKCU\SessionInformation key does not exist under the Default Users HKCU hive. Attempts to modify an HKCU key that exists for a logged on user but does not exist for the Default User always results in this error.

Though a user's HKCU hive cannot be modified as Local System, HKEY_USERS (HKU) hive can be accessed. Each user's HKCU hive is represented with a Security ID (SID) such as "S-1-5-18". Any changes made under a SID key in this hive occurs to the user's HKCU hive associated with the SID.

Understanding the ERRORLEVEL Variable

While a batch file runs, a dynamic environment variable called ERRORLEVEL is set to the return value of each application that is called. Whatever ERRORLEVEL is set to when the batch file terminates is the ERRORLEVEL passed back to the Core Server.

The Core Server shows a success for an ERRORLEVEL 0, but fails if ERRORLEVEL is set to any value other than 0.

Writing Batch Files to Return ERRORLEVEL 0 on Success

When pushing batch, the only return code that shows successful on the Core Server is 0. Batch files must be developed so that the ERRORLEVEL variable equals 0 when the batch file runs successfully.

Example of an Incorrectly Designed Batch File

Below is an example of a batch file which is incorrectly designed. This batch file could fail and still show as successful in the LANDesk console.

```
1. @ECHO ON
2. copy \\server\share\install.exe c:\temp\install.exe
3. copy \\server\share\install.exe c:\temp\app.dll
4. c:\temp\install.exe
5. reg add hklm\software\app
```

This seems like a simple enough batch file. It copies two files from a share and then adds a registry key. However, it is flawed.

Line 1 (@ECHO ON) does not modify %ERRORLEVEL% so it can be ignored.

Line 2 could fail for many reasons: the file is not found, the server is not accessible, or the share permissions do not allow access, or many other reasons. Such a failure causes the copy command to return a failure code. In the case where the share permissions are incorrect, the copy command returns 1, meaning the copy failed because access is denied. The ERRORLEVEL variable is now set to 1.

Line 3 calls the same share and also fails, again setting ERRORLEVEL to 1.

Line 4 fails because there is not an install.exe in c:\temp due to the failure in line 2. This results in setting ERRORLEVEL to 9009.

Line 5 calls the reg.exe command. This command is available in the system32 directory which is in the PATH on Windows® XP. The share is not accessed so there is nothing to keep this command from succeeding. The registry key is successful added, and ERRORLEVEL is now set to 0.

The batch file ends with ERRORLEVEL set to 0. This result is passed back to the Core Server and the job is marked as successful even though every line but the last one failed. This batch file is designed incorrectly. LANDesk only accesses the final result of the batch file. It is the responsibility of the batch file developer to make sure that each line succeeds and that the batch file ends with the proper return code.

Example of a Correctly Designed Batch File

Below is an example of a batch file correctly designed to make sure that each line completes successfully:

```
1. @ECHO ON
2. copy \\server\share\install.exe c:\temp\install.exe
3. IF NOT "%ERRORLEVEL%"=="0" GOTO end
4. copy \\server\share\install.exe c:\temp\app.dll
5. IF NOT "%ERRORLEVEL%"=="0" GOTO end
6. c:\temp\install.exe
7. IF NOT "%ERRORLEVEL%"=="0" GOTO end
8. reg add hklm\software\app
9. :end
```

In this batch file and IF statement is used after each command to verify that the command succeeded and if the command fails it uses the GOTO command to send it to the "end" label where the batch file ends.

This batch file could also be written as follows:

```
1. @ECHO ON
2. copy \\server\share\install.exe c:\temp\install.exe
3. IF NOT "%ERRORLEVEL%"=="0" EXIT /B %ERRORLEVEL%
4. copy \\server\share\install.exe c:\temp\app.dll
5. IF NOT "%ERRORLEVEL%"=="0" EXIT /B %ERRORLEVEL%
6. c:\temp\install.exe
7. IF NOT "%ERRORLEVEL%"=="0" EXIT /B %ERRORLEVEL%
8. reg add hklm\software\app
9. :end
```

In this batch file the IF state is also used. It calls the EXIT command. It uses the /B switch which means terminate the batch file and it passes the ERRORLEVEL variable to the EXIT command so the batch is terminated with the current ERRORLEVEL.

It is important that the following syntax is used when calling the EXIT command:

```
EXIT /B [Exit Code]
```

Calling the EXIT command without a /B terminates the batch process and the process that launched the batch file. This causes the LANDesk client to report a failure back to the Core Server.

For more information see the following site:
http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/ServerHelp/fdc12a63-df4e-49e7-94d6-177536b18eb6.mspx

Handling Success Codes Other Than 0

As stated previously, batch files must be designed to return 0 when successful. However, many applications return success codes other than 0. When this is the case, it is important to design the batch file to check for the proper success code and to make sure that the ERRORLEVEL variable is set to 0 when a batch file terminates successfully.

Many simple batch files are not going work when running applications that have a success code other than 0. To demonstrate this, review the example of the basic batch file shown below.

```
1. @ECHO ON
2. :: Batch file 1 is written to copy a file
3. copy c:\file1 c:\file2
```

Using the basic batch file, the ERRORLEVEL variable is always set by the last command. In the above example, a success code for the copy command is 0. However, look at the example below:

```
1. @ECHO ON
2. :: Batch file 1 is written to copy a file
3. LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\Ldscn32.exe" /taskid=1001
   /cmd="/NTT=Core:5007 /S=Core /I=HTTP://Core/ldlogon/ldappl3.ldz"
   /freq=3600
```

In this example, LANDesk's localsch.exe is called to add a Local Scheduler task. When Local Scheduler succeeds in adding a task, the return code is not 0, but instead is the Task ID. In this case, 1001 is the success code.

In this instance, it is impossible to use the basic batch file to successfully pass a 0 status back to Core Server. A complex batch file must be used to make sure the Local Scheduler task is added successfully and the batch file completes with ERRORLEVEL 0.

Note: Notice that the %LDMS_LOCAL_DIR:~0,5% to get the path to ldclient.

This batch file can be written in any of the following three ways:

Batch file 1

```
1. @ECHO ON
2. :: Batch file 1  - written to add a local scheduler task
3. LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\Ldscn32.exe" /taskid=1001
   /cmd="/NTT=Core:5007 /S=Core /I=HTTP://Core/ldlogon/ldappl3.ldz"
   /freq=3600
4. IF "%ERRORLEVEL%"=="1001" CMD /C
```

Batch file 1 checks to makes sure that line 3 succeeded and if so, it runs CMD /C which is always successful and returns 0. The use of CMD /C is a common method for setting ERRORLEVEL back to 0.

Batch file 2

```
1. @ECHO ON
2. :: Batch file 2  - written to add a local scheduler task
3. LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\Ldscn32.exe" /taskid=1001
   /cmd="/NTT=Core:5007 /S=Core  /I=HTTP://Core/ldlogon/ldappl3.ldz"
   /freq=3600
4. IF "%ERRORLEVEL%"=="1001" EXIT /B 0
```

Batch file 2 uses the EXIT command to terminate the batch file and since the exit code is explicitly defined as 0, the batch file ends with ERRORLEVEL 0 and the Core Server knows it is successful.

Batch file 3

```
1. @ECHO ON
2. :: Batch file 3  - written to add a local scheduler task
3. LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\Ldscn32.exe" /taskid=1001
   /cmd="/NTT=Core:5007 /S=Core  /I=HTTP://Core/ldlogon/ldappl3.ldz"
   /freq=3600
4. IF "%ERRORLEVEL%"=="1001" SET ERRORLEVEL=0
```

Batch file 3 statically sets the ERRORLEVEL variable to 0 so when the batch file ends, it is successful.

This command works for one command, but when running multiple commands, it may not have the desired results. Consider the following batch file:

```
1. @ECHO ON
2. :: Batch file 1 is written to copy a file
3. LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\Ldscn32.exe" /taskid=1001
   /cmd="/NTT=Core:5007 /S=Core  /I=HTTP://Core/ldlogon/ldappl3.ldz"
   /freq=3600
4. IF "%ERRORLEVEL%"=="1001" SET ERRORLEVEL=0
5. LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\amclient.exe" /taskid=1002
   /cmd="/apm /s /c" /freq=3600
6. IF "%ERRORLEVEL%"=="1002" SET ERRORLEVEL=0
```

This batch file may not perform as at first intended. If line 3 succeeds, line 4 statically sets ERRORLEVEL to 0. Examine what is going to happen after line 5 runs and the ERRORLEVEL variable is checked on line 6. ERRORLEVEL is statically set to 0, so line 6 reports to the console that the batch file has succeeded regardless of whether line 5 succeeds or fails. This could cause incorrect status. This can easily be resolved by using the EXIT command instead of statically setting ERRORLEVLE to 0. This is demonstrated in the following example.

```
1. @ECHO ON
2. :: Batch file 1 is written to copy a file
3. LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\Ldscn32.exe" /taskid=1001
   /cmd="/NTT=Core:5007 /S=Core  /I=HTTP://Core/ldlogon/ldappl3.ldz"
   /freq=3600
4. IF NOT "%ERRORLEVEL%"=="1001" EXIT /B %ERRORLEVEL% ELSE CMD /C
5. LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\amclient.exe" /taskid=1002
   /cmd="/apm /s /c" /freq=3600
6. IF "%ERRORLEVEL%"=="1002" EXIT /B 0
```

This batch design works properly because line 4 now checks the value of ERRORLEVEL and if it is not 1001, EXIT is called, terminating the batch file.

Also note that if ERRORLEVEL does equal 1001, instead of statically setting ERRORLEVEL to 0, CMD /C is run. This results in ERRORLEVEL being set to 0 without statically setting it.

Ignoring Insignificant Failures

Often a line can fail and it does not matter. However such failures may set ERRORLEVEL to a value other than 0 resulting in LANDesk Console reporting a failure code.

Consider the following batch file:

```
1. @ECHO ON
2. :: Batch file 1 is written to make a logs directory
3. MKDIR “c:\logs”
```

In this batch file the goal is to create a directory on a client where logs can be created and stored. This batch file is deployed to a group of test machines and it works successfully; the c:\logs directory is created on the test clients.

Consider what would happen if the next day the script is deployed to all clients. Since the directory already exists on the test clients, the make directory command would fail with the following error:

```
A subdirectory or file c:\logs already exists.
```

The MKDIR command fails with a return code of 1. Only 0 is a successful return code for a batch file. In order to make this successful, similar methods used to handle non-zero success codes can be used. Any of the following four batch files accomplish this:

```
1. @ECHO ON
2. :: Batch file 1 This batch creates the c:\logs directory
3. MKDIR “c:\logs”
4. EXIT /B 0
```

```
1. @ECHO ON
2. :: Batch file 2 - This batch creates the c:\logs directory
3. MKDIR “c:\logs”
4. cmd /c
```

These first two batch files simply set ERRORLEVEL to 0 after the running line 3, so the result of line 3 is ignored. This could cause a problem if the directory fails to create do to a different error such as “Access denied”.

Avoiding Insignificant Failures

It is often better to avoid possible failures than to ignore them. Observe the following batch file.

```
1. @ECHO ON
2. :: Batch file 4 – This batch creates the c:\logs directory
3. IF EXIST “c:\logs” EXIT /B 0 ELSE (MKDIR “c:\logs”)
```

This batch files demonstrates the use of an IF statement to verify that the folder is created. This is a more efficient than ignoring a failure.

Using SDCLIENT.EXE in a Batch file

SDCLIENT.EXE is the tool that launches that batch file. When running a batch file, SDCLIENT.EXE currently cannot be used in a batch file to install an MSI, an executable, or another batch file. Nor can any parameters that exist for these install types be used.

Note: Since only one install can occur at a time, it should be obvious that a batch file cannot run AMCLIENT.EXE to install policies.

SDCLIENT.EXE can be used to send messages to the Core Server and to handle reboots. It may be used to download a file, but any command-line web utility would probably be easier to use and have more desirable features.

Sending Messages to the Core Server

LANDesk's sdclient.exe file can be called from a batch file to send status back to the scheduled task on the Core Server. This updates the messages column under the scheduled task for the device.

Syntax

The syntax for sending a message to the Core Server is as follows:

```
sdclient.exe /msg="text"
```

How Messages Are Sent

Messages are sent using the same method as Policy status, by sending an XML file through an HTTP post to the following web application:
http://coreserver/incomingdatra/postcgi.exe?prefix=sdstatus\&suffix=.swd.xml

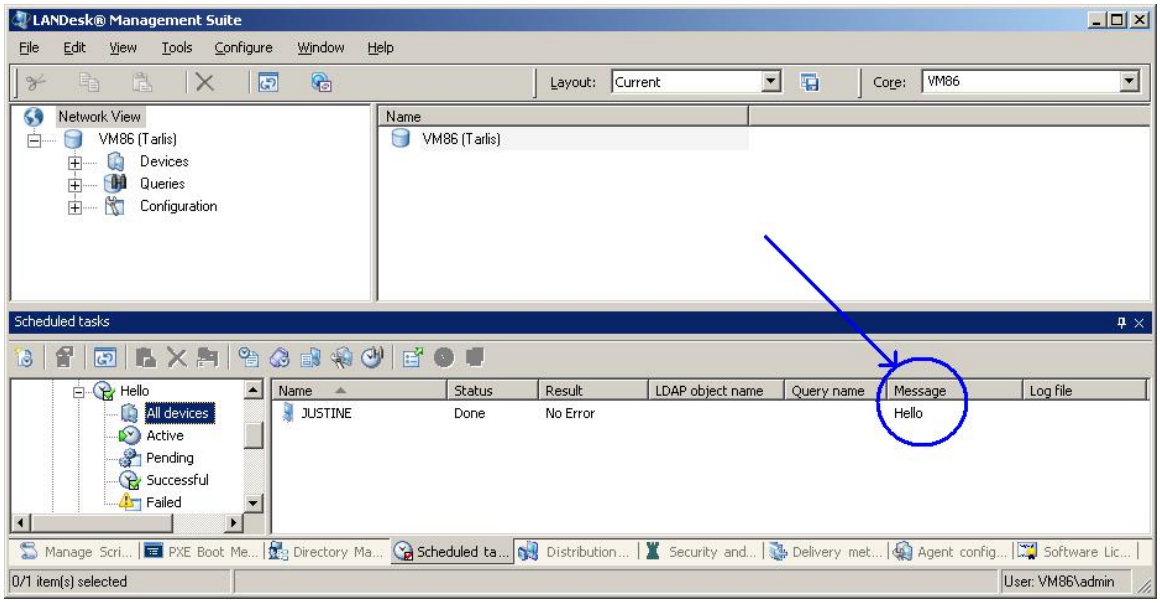
Warning: Modifications to the permissions of the incoming folder or the postcgi.exe file could prevent messages from returning due to access denied errors on this web application. Also note that modifications to the either of the following group policies can also prevent messages from returning due to access denied errors: (1) Adjust memory quotas for a process; (2) Replace a process level token.

Example

Below is an example of a batch file that uses this feature. This batch file adds a local scheduler task and sends the result back in a message to the scheduled task on the Core Server.

```
1.  @ECHO ON
2.  :: Hello.bat – Sending messages to core
3.  sdclient.exe /msg="Hello"
```

In this batch file, a message is sent on line 3 to the console saying simply "Hello". The result in the LANDesk Console is as follows:



Normally this message would be used in a more informational fashion. Below is an example of a batch file that uses the message feature to show what command the batch is currently processing.

```
1.  @ECHO ON
2.  :: This batch file runs multiple commands
3.  sdclient.exe /msg="Running Command 1"
4.  \\server\share\command1.exe
5.  IF NOT "%ERRORLEVEL%"=="0" EXIT /B %ERRORLEVEL%
6.  sdclient.exe /msg="Command 1 success.  Running Command 2"
7.  \\server\share\command2.exe
8.  IF NOT "%ERRORLEVEL%"=="0" EXIT /B %ERRORLEVEL%
9.  sdclient.exe /msg="Command 2 success.  Running Command 3"
10. \\server\share\command3.exe
11. IF NOT "%ERRORLEVEL%"=="0" EXIT /B %ERRORLEVEL%
12. sdclient.exe /msg="All commands completed successfully"
13. EXIT /B 0
```

In this batch file, each command is preceded by a message to the core. A quick look at the message section of a scheduled task displays which command the batch file is currently running.

Sending ERRORLEVEL in the Message

It is often beneficial to send the result code of a command in a message to the Core Server. When running sdclient.exe to send a message to the Core Server, ERRORLEVEL is set to 0. This could result in a failed batch file appearing to be successful. To demonstrate this, look at the batch file below:

```
1. @ECHO ON
2. :: This batch file sends ERRORLEVEL to the Core Server.
3. :: Run sdclient.exe to result in ERRORLEVEL being 0
4. \\server\share\command1.exe
```

```
5. IF "%ERRORLEVEL%"=="0" ( sdclient.exe /msg="Command 1 completed with
   result code %ERRORLEVEL%."
6. ) ELSE ( sdclient.exe /msg="Command 1 failed with result code
   %ERRORLEVEL%.  Exiting" )
7. :end
```

If command1.exe fails, the IF statement calls sdclient.exe to send the failure message to the Core Server. Sdclient.exe succeeds, setting ERRORLEVEL to 0 and the batch file ends. This causes the batch file to appear successful. To resolve this, assign ERRORLEVEL to a static variable after running command1.exe and pass that variable to the EXIT command.

In the batch file below, the value of ERRORLEVEL is assigned to the variable called ResultCode and the batch file ends with the value of ResultCode.

```
1. @ECHO ON
2. :: This batch file sends ERRORLEVEL to the Core Server.
3. :: Run sdclient.exe to result in ERRORLEVEL being 0
4. \\server\share\command1.exe
5. :: ERRORLEVEL is stored as a static variable
6. SET ResultCode =%ERRORLEVEL%
7. IF "%ResultCode%"=="0" ( sdclient.exe /msg="Command 1 completed with
   result code %ResultCode%."
8. ) ELSE ( sdclient.exe /msg="Command 1 failed with result code
   %ResultCode%.  Exiting" )
9. :end
10.  EXIT /B %ResultCode%
```

In this batch file if command1.exe fails, the failure code is assigned to a static variable, which is passed to the EXIT command, allowing the batch file to end with the proper failure code.

Handling Batch File Reboots

Batch files can handle reboots using a combination of LANDesk's sdclient.exe and batch file parameters. Familiarity with how batch file parameters are used is required to understand this section. See the following site for more information on using parameters in batch files:
http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/ServerHelp/fdc12a63-df4e-49e7-94d6-177536b18eb6.mspx

Syntax

The two line syntax for handling a reboot in a batch file using sdclient.exe is as follows:

```
sdclient.exe /onreboot /bat /p=%0 /cmds=<parameter>
sdclient.exe /reboot
```

The first command tells sdclient.exe what batch file to run on reboot and what parameter to pass to this batch file. The /p=<batch file> parameter tells sdclient.exe what batch file to run after the reboot and %0 is a standard batch file variable that resolves to the currently running batch file. The /cmds=<parameter> is the

parameter to pass to the batch file after reboot. This parameter is used to skip the section of the batch file already completed before the reboot and start at a different label.

Example

The batch file below is an example of how to use sdclient.exe to reboot and how to use batch file parameters to run only the post-reboot tasks after the reboot occurs.

```
1.  @ECHO ON
2.  :: This batch file runs pre-reboot tasks, reboots, and
3.  :: then runs post-reboot tasks.
4.
5.  :checkparameter
6.  :: Batch fails if two parameters are given or if the first
7.  :: parameter is does not match a valid label.
8.  IF NOT "%2."="." GOTO invalidparameter
9.  IF "%1."="." GOTO pre-reboot
10. IF "%1"="post-reboot" GOTO %1
11.
12. :invalidparameter
13. :: Only one parameter is allowed.
14. ECHO Error: Invalid Parameter
15. ECHO Valid parameter: [post-reboot]
16. GOTO end
17.
18. :pre-reboot
19. sdclient.exe /msg="Running pre-reboot tasks."
20. ::some pre-reboot tasks happen here
21. IF NOT "%ERRORLEVEL%"="0" GOTO end
22.
23. :reboot
24. sdclient.exe /msg="Rebooting."
25. sdclient.exe /onreboot /bat /p=%0 /cmds=post-reboot
26. sdclient.exe /reboot
27. GOTO end
28.
29. :post-reboot
30. sdclient.exe /msg="Running post-reboot tasks."
31. ::some post-reboot tasks happen here
32. IF NOT "%ERRORLEVEL%"="0" GOTO end
33.
34. :end
```

This batch file has multiple sections marked with labels including: checkparameter, invalidparameter, pre-reboot, reboot, post-reboot, and end.

The first two sections, checkparameter and invalidparameter, are checking to make sure the batch file parameter is valid. If no parameter is given, as is the case when the batch file is first launched, line 9 tells the batch file to skip to the pre-reboot label on line 18.

Lines 18-21 run the pre-reboot tasks. If it fails, the batch file ends sending a failure back to the core. If successful the pre-reboot tasks are followed by the reboot lines 23-27.

The reboot task in lines 25-29 tells sdclient.exe to launch the batch file after the reboot using “post-reboot” as a parameter. Once the reboot occurs, sdclient.exe launches the batch file with “post-reboot” as a parameter.

Line 10 checks that proper parameter is used, so it sends processing to the post-reboot label on line 29. This skips the pre-reboot section and the reboot section. The post-reboot tasks occur in lines 29-33 and if successful, the batch file ends successful.

Warning: Setting the Delivery Method’s Reboot option to Never Reboot will prevent the batch file from rebooting.

Passing Inventory Data as a Parameter

LANDesk has the ability to use database macros to pass data that is already gathered into the inventory database to a batch file as a parameter. This section describes how this can be done.

Often certain batch file tasks may need to occur based on a dynamic value that does not exist in the client as a variable.

Familiarity with how batch file parameters are used is required to understand this section. See the following site for more information on using parameters in batch files:
http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/ServerHelp/fdc12a63-df4e-49e7-94d6-177536b18eb6.mspx

Understanding Inventory

LANDesk stores accesses inventory in the database in a hierarchical fashion. Everything exists under the main heading "Computer". Under the main heading, there are multiple subheadings. Under any heading there can be either more subheadings or there can be one or more data entries assigned a value. For example, under the main heading, "Computer", there are many subheadings, such as "Network" or "OS"; and there are many data entries with assigned values, such as "Device ID" or "Device Name". Under the "Network" subheading exists other subheadings, such as "TCPIP"; and there are other data entries (attributes) with assigned values such as "NIC Address". This can be best understood by browsing the inventory of a machine in the database using the LANDesk Console.

Database Macro Syntax

The following is the proper syntax for a database macro.

```
%Computer.[Subheading.Subheading.]Data%
```

The macro begins with a % symbol. The main heading "Computer" is required and the subheadings may be optional. The Data field is required. The macro then is terminated with an ending % symbol.

It is the assigned value of a data field that is passed to a batch file, the headings and data names are not passed. The syntax for the database macros must call a data name with an assigned value. It is the value that is returned from a database macro not a heading name or data name.

To demonstrate this, the following example is a macro that gets the computer name from the database:

```
%Computer.Device Name%
```

Notice it calls the required primary heading "Computer" and data name "Device Name" but does not call a subheading.

Note: The example above, %Computer.Device Name%, would probably never be used as there is already an environment variable, %Computername%, on the client computer that can be accessed with a batch file. Data easily accessible on the client should be preferred over data from the database. This simplifies the batch file and extra processing and database access needed on the Core Server to evaluate a macro.

The following macro includes two subheadings. This macro gets from the database the total physical memory in bytes of a client.

%Computer.Memory.Physical.Bytes Total%

This syntax is the same syntax that can be seen in a Query on the LANDesk Console. Each element in the macro can be quoted as seen in a query but quotes are normally not used because the entire macro is preceded and ended with a % symbol.

Note: Unlike getting the computer name, there does not already exist an easy way to access the total bytes of physical memory with a batch file. Passing this value from the database as a parameter is more appealing than having to run and parse the output of a third party memory tool.

Batch File Parameter Syntax

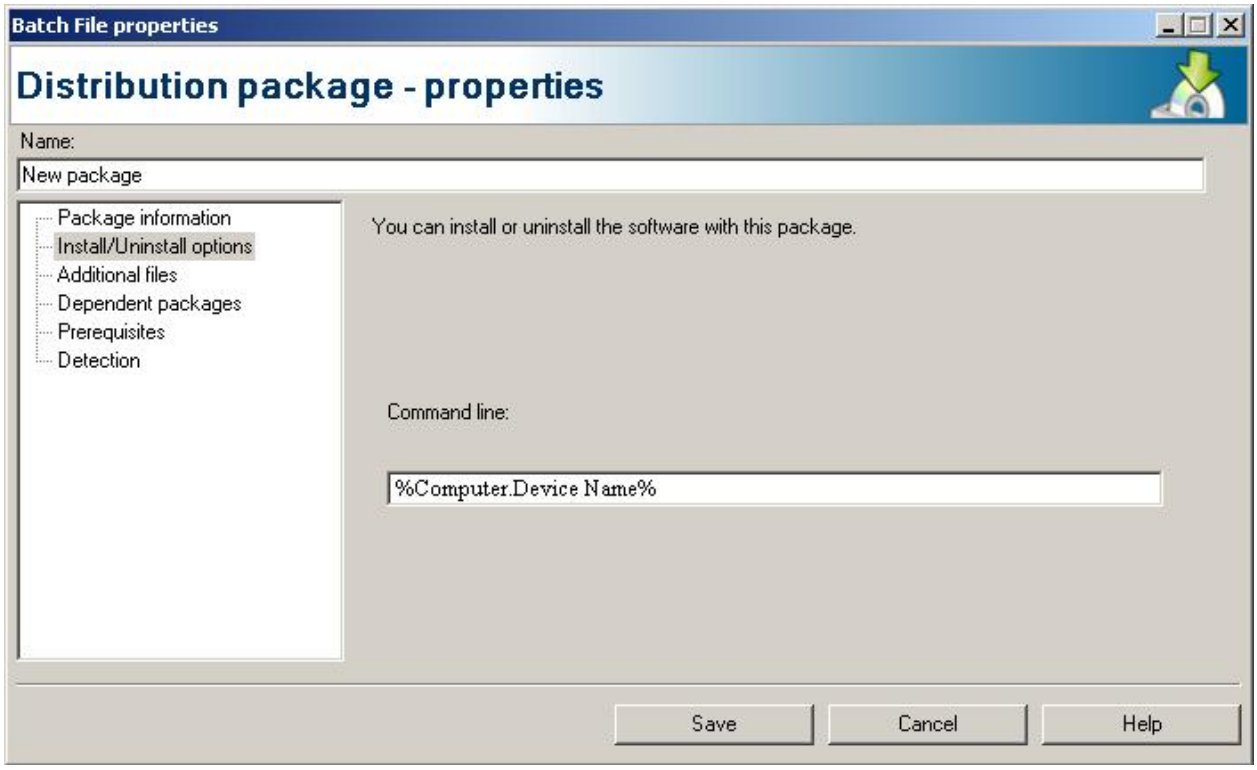
When passing a parameter to a batch file, the syntax in a command prompt is:

```
C:\path\to\my\batch.bat [param1 | param2 | param3 |... param9]
```

The first parameter passed can be accessed by a batch file using the variable %1, the second parameter using %2, and third %3, to the ninth parameter as %9. If more than nine parameters are used, SHIFT will have to be used inside the batch file. See the following Microsoft site for more information on SHIFT:
http://technet2.microsoft.com/WindowsServer/en/Library/fdc12a63-df4e-49e7-94d6-177536b18eb61033.mspx

LANDesk's database macros cannot be accessed from the command line. With LANDesk Management Suite a Batch File Distribution Package is created which points to a batch file. When creating a Distribution Package in the LANDesk Console, the Install/Uninstall Options screen is where batch file parameters are entered.

Database macros entered into the Command Line Parameters field are evaluated on the Core Server on a per machine basis. The Core Server replaces the macro with the actual value before the parameter is passed to the client. For example, if the macro being called is %Computer.Network.NIC Address% and the value in the database is 000AE43531EC, the parameter passed to the batch file is 000AE43531EC. If the macro being called is %Computer.Device Name% and the value in the database is Computer100, the parameter passed to the batch file is Computer100.



Handling Spaces

If the parameter has a space then it must be quoted otherwise it will be treated as multiple parameters. To demonstrator this, imagine the macro being called passes the OS name. The OS Name value can contain multiple spaces. If it were passed to a Batch File Distribution Package as follows:

```
%Computer.OS.Name%
```

One possible return value could be *Microsoft Windows XP Professional*. Without quotes, the first parameter (%1) would be *Microsoft*, the second parameter (%2) would be *Windows*, and the third parameter (%3) would be *XP*, the fourth parameter (%4) would be *Professional*. This may not be what is desired. Surrounding the macro in quotes allows the entire value to be included in the first parameter (%1).

"%Computer.OS.Name%"

However, remember that the parameter passed includes the quotes. This means the first parameter (%1) is not simply the string *Microsoft Windows XP Professional*; the parameter is actually *"Microsoft Windows XP Professional"*, including the surrounding quotes along with the string.

Also not that a quote was is not required if the variable has a space, only if the value contains a space.

Installing an SWD Package from a Batch File

SWD packages are single file executables created with Package builder. It is recommended that any SWD package be installed as an SWD Distribution Package. However, there may be a time when installing and SWD package from a batch file is desired.

SWD packages are installed with an installer called INST32.EXE, which is located in the ldclient directory on an Agent workstation. To install an SWD package in a batch file, use inst32.exe switches. Common switches are listed. For a complete list of switches, read the *Using Custom Scripts* whitepaper.

/Ac This parameter indicates that the installation cannot be cancelled.

/All The uninstall all command.

/An Completely disables INST32's UI. It is not necessary to specify other UI-controlling commands because the user will never be prompted for any input when in this mode. Also, no installation progress will be displayed.

/Ah+ If a package is being installed for the first time, this command is ignored. If, however, the package has been installed previously, this will tell the installer to automatically heal the package without prompting the user.

/Ah- If a package is being installed for the first time, this command is ignored. If, however the package has been installed previously, this will tell the installer to automatically force a reinstall of the package without prompting the user.

/Ai It causes INST32 to refrain from rebooting when completing an install and instead to return a reboot request code. Some informational dialogs that require the user to click OK are removed when in this mode.

/Ao This is a "run once" mode kind of switch, which is used when applying the restore system state package on the build computer. It stops the installer from rebooting the computer in order to complete an installation.

The following example demonstrates a batch file that launches an SWD package.

```
1. @ECHO ON
2. :: This batch installs an SWD package silently and disallows any
3. :: reboots, allowing for the batch file to handle them.
4.
5. SWDpackage.exe /Ai /Ao /An /Ah-
```

Line 5 could also be written as follows:

```
5. inst32.exe SWDpackage.exe /Ai /Ao /An /Ah-
```

LANDesk Support

LANDesk does not support the content inside a batch file, whoever wrote the batch file must provide that support. LANDesk does not train on how to use batch files.

LANDesk supports deploying and launching a batch file. This support includes downloading the batch file (and all additional files if applicable) to client workstations and then running the batch file. If the batch file returns an error, it is the responsibility of the batch file developer to determine which line is the batch file causes this error and resolve it.

If a support case is opened with LANDesk, as soon as LANDesk launches the batch file is launched, LANDesk is determined to be working properly and the case will be closed. It is the sole responsibility of the batch file to developer to troubleshoot errors that occur within a batch file. It is the batch file developer's responsibility to make sure this batch file works when run as Local System. It is the batch file developer's responsibility to troubleshoot.

If a third party application is launched within a batch file and this applications is failing, the issue must be address with the third party company and not with LANDesk support.

The only exception to this is when the line failing in the batch file properly calls a LANDesk utility such as SDCLIENT.EXE or LOCALSCH.EXE and those LANDesk files are failing.

Troubleshooting

Troubleshooting batch files can be complex for many reasons. Each batch file is unique and calls a variety of commands. The majority of problems are due the batch file itself and the way it is written. Most errors are avoided if the batch file is correctly tested to work as Local System. (See the section titled Understanding the Microsoft Local System Account found previously in this document).

When errors do occur, it is important to understand what information is available for troubleshooting them. This section provides information on how to troubleshoot error messages.

The steps to troubleshooting a batch file are as follows:

- Step 1 - Obtain the Log Files
- Step 2 - Obtain the Failure Code
- Step 3 - Find the Line in the Batch File That Failed
- Step 4 – Find the Reason the Line Failed

Step 1 - Obtaining the Log Files

There are two main logs to obtain. The Core Server's task log and the client's task log.

Scheduled Task Log on the Core Server

The Core Server creates a log for the batch file task and stores that log in a directory simply named "log" located in the ManagementSuite directory, or the ldlog share. This log is named ScheduledTaskHandler_#.log (where # represents the scheduled task identifier from the database).

When a scheduled task fails, the Core Server copies the client's task log and makes it available through the console by right clicking on a failed client in a task and choosing View log file.

Scheduled Task Log on the Client

The client creates a log for the batch file task and stores that log in the directory called "data" located in the LANDesk\LDClient directory (usually this directory is C:\Program Files\LANDesk\LDClient\data). This log is named sdclient_task#.log (where # represents the scheduled task identifier from the database). This log is usually available from the console on a failed task and contains any data passed to STDOUT by the batch file.

Step 2 – Obtaining the Failure Code

The result code is located in both the ScheduledTaskHandler_#.log on the Core Server and the sdclient_task#.log on the client. The result code matches the final value of ERRORLEVEL when the batch file terminates.

A success code looks as follows:

```
Wed, 07 Dec 2005 15:51:32 process of package is complete, result 0
(0x00000000 – code 0)
```

A failure code is very similar, ending with a number other than 0:

```
Wed, 07 Dec 2005 15:51:32 process of package is complete, result
-2147024893 (0x8007002 – code 1)
```

NOTE: The failure string displayed in the task may be incorrect as batch files are linked to facility 7, Microsoft, which results in the Win32 strings being displayed on the console regardless of what command in the batch file really returned the error. Except when using the basic batch file to deploy an MSI file or other Win32® tasks, the failure string on the console should be ignored.

Step 3 - Finding the Line in the Batch File That Failed

Once the failure code has been determined, the next step in troubleshooting is to determine which line in the batch file returned the failure code. As batch file only fails when the ERRORLEVEL variable is set to something other than 0 when it completes, any failure indicates that one or more of the lines in the batch file are setting ERRORLEVEL to something other than 0. This can only be resolved by finding the line or lines that are failing. Once those failures are resolved, the batch file continues.

Making Sure ECHO Is Turned On

When troubleshooting, it is beneficial to make sure that ECHO is turned on. Only the information passed to STDOUT is found in the sdclient_task#.log file.

When a batch file is called, anything passed to STDOUT is displayed in the sdclient_task#.log file. (For more information on STDOUT, see Using command redirection operators on the MSDN® web site.) If ECHO is off, the commands themselves are not passed to STDOUT. Commands may or may not send results to standard out. If ECHO is on, every line in the batch file, as well as anything the command passes to STDOUT, are included in the log.

To demonstrate this, the batch file listed below is used with both ECHO on and ECHO off. This batch file runs MKDIR to make a directory that already exists. With ECHO ON This would result in the log file also listed below:

Batch file – MKDIR.bat with ECHO turned off

```
1. @ECHO OFF
2. ::This batch file runs MKDIR and has ECHO turned off.
3. MKDIR c:\temp
4. SET result=%ERRORLEVEL%
5. sdclient.exe /msg="RESULT = %result%"
6. EXIT /B %result%
```

Log File – sdclient_task#.log from MKDIR.bat with ECHO turned off

```
Wed, 21 Dec 2005 11:20:17 File (http://vm86/share/batch/MKDIR.bat) is not in cache
Wed, 21 Dec 2005 11:20:18 WM_CREATE
Wed, 21 Dec 2005 11:20:18 WM_STARTCLIENT
Wed, 21 Dec 2005 11:20:20 Downloading file 1 of 1 from 'http://vm86/share/batch/MKDIR.bat'
Wed, 21 Dec 2005 11:20:20 WinNT
Wed, 21 Dec 2005 11:20:20 Batch file Client Thread
Wed, 21 Dec 2005 11:20:20 PackagePath:   [http://vm86/share/batch/MKDIR.bat]
Wed, 21 Dec 2005 11:20:21 Custom Message : RESULT = 1
Wed, 21 Dec 2005 11:20:21 Core name 'vm86:172.16.27.38' obtained from active task list
Wed, 21 Dec 2005 11:20:21 Setting status for batch file to 1
Wed, 21 Dec 2005 11:20:21 Batch file status of 1 obtained from shared memory
Wed, 21 Dec 2005 11:20:21 treating status as win32 HRESULT, updated value 8007000108x
Wed, 21 Dec 2005 11:20:21 Bat file output :

C:\Program Files\LANDesk\LDClient\sdmcache\share\batch>call "MKDIR.bat"

Wed, 21 Dec 2005 11:20:21 Installation result 80070001Wed, 21 Dec 2005 11:20:21 processing of
package is complete, result -2147024895 (0x80070001 - code 1)
```

Batch file – MKDIR.bat with ECHO turned on

```
1. @ECHO ON
2. ::This batch file runs MKDIR and has ECHO turned on.
3. MKDIR c:\temp
4. SET result=%ERRORLEVEL%
5. sdclient.exe /msg="RESULT = %result%"
6. EXIT /B %result%
```

Log File – sdclient_task#.log from MKDIR.bat with ECHO turned on

```
Wed, 21 Dec 2005 11:20:17 File (http://vm86/share/batch/MKDIR.bat) is not in cache
Wed, 21 Dec 2005 11:20:18 WM_CREATE
Wed, 21 Dec 2005 11:20:18 WM_STARTCLIENT
Wed, 21 Dec 2005 11:20:20 Downloading file 1 of 1 from 'http://vm86/share/batch/MKDIR.bat'
Wed, 21 Dec 2005 11:20:20 WinNT
Wed, 21 Dec 2005 11:20:20 Batch file Client Thread
Wed, 21 Dec 2005 11:20:20 PackagePath:   [http://vm86/share/batch/MKDIR.bat]
Wed, 21 Dec 2005 11:20:21 Custom Message : RESULT = 1
Wed, 21 Dec 2005 11:20:21 Core name 'vm86:172.16.27.38' obtained from active task list
Wed, 21 Dec 2005 11:20:21 Setting status for batch file to 1
Wed, 21 Dec 2005 11:20:21 Batch file status of 1 obtained from shared memory
Wed, 21 Dec 2005 11:20:21 treating status as win32 HRESULT, updated value 8007000108x
Wed, 21 Dec 2005 11:20:21 Bat file output :
C:\Program Files\LANDesk\LDClient\sdmcache\share\batch>call "MKDIR.bat"

C:\Program Files\LANDesk\LDClient\sdmcache\share\batch>MKDIR c:\temp

C:\Program Files\LANDesk\LDClient\sdmcache\share\batch>SET result=1

C:\Program Files\LANDesk\LDClient\sdmcache\share\batch>"c:\program
files\LANDesk\LDClient\sdclient.exe" /msg="RESULT = 1"

C:\Program Files\LANDesk\LDClient\sdmcache\share\batch>EXIT /B 1

C:\Program Files\LANDesk\LDClient\sdmcache\share\batch>"C:\Program
Files\LANDesk\LDClient\sdclient.exe" /setbatchstatus=1

Wed, 21 Dec 2005 11:20:21 Installation result 80070001
Wed, 21 Dec 2005 11:20:21 processing of package is complete, result -2147024895 (0x80070001 -
code 1)
```

Notice that with ECHO turned on, every line in the batch file (marked in red) is included in the log file. This is useful in detecting a failed command.

Echoing the ERRORLEVEL after Each Command

A simple way to find what ERRORLEVEL is set to after running each command is to add a line that echoes the ERRORLEVEL.

Example:

```
1. @ECHO ON
2. ::This batch runs two commands and echoes ERRORLEVEL of each
3. MKDIR c:\temp
4. ECHO %ERRORLEVEL%
5. cacls c:\temp /E /P BUILTIN\users:N
6. ECHO %ERRORLEVEL%
```

The above batch file echoes the result code of each command to the log. This batch file can be enhanced to clarify the echoed information.

```
1. @ECHO ON
2. ::This batch file runs a command and echoes the ERRORLEVEL
3. MKDIR c:\temp
4. ECHO Command 1 – Result = %ERRORLEVEL%
5. cacls c:\temp /E / BUILTIN\users:N
6. ECHO Command 2 – Result = %ERRORLEVEL%
```

Step 4 - Finding the Reason the Line Fails

Each line in a batch file is unique and can launch many types of processes including shell commands, executables, MSIs, VBScripts, other batch files, and more. When a batch file fails and the failed line is discovered, the troubleshooting now involves that line. To troubleshoot this line, determine the following:

- What command is this line using?
- What company is responsible for this command and what documentation can this company provide me for this command?
- What is this line supposed to do?
- Does this command have switches and are they used properly?
- What error message is the command returning?

Example – Troubleshooting a Failed Line

To demonstrate this method of troubleshooting, look at the following basic batch file:

```
1. @ECHO ON
2. ::This batch file displays the run keys from the registry
3. reg.exe query HKLM\Software\Microsoft\Windows\CurrentVersion\Run
```

After deploying this batch file, this command fails on all Windows® 2000 machines. To troubleshoot the failure, the above questions can be answered. The answered can then be evaluated to determine the cause of the failure.

The following are the answers to the above troubleshooting questions for this problem with reg.exe on Windows® 2000:

- This line uses the reg.exe command
- Reg.exe is made by Microsoft® and searching Microsoft's site results in much documentation.
- This command has switches which can be seen by running reg.exe /? at a command prompt.
- After running the command manually at a Local System command, the following error message is returned:

  ```
  'reg.exe' is not recognized as an internal or external command,
  operable program or batch file.
  ```

By answering the above questions it is determined that Windows® 2000 does include reg.exe by default while Windows XP does. Reg.exe must be added to Windows® 2000.

Capturing STDERR in the Log File

Often the workstation on which a batch file failed cannot be accessed. This means that the only available method of troubleshooting is the log file. Many commands output error messages to STDERR. Since only STDOUT is captured in the sdclient_task#.log, STDERR is not captured by default.

```
1. @ECHO ON
```

```
2. ::This batch file displays the run keys from the registry
3. MKDIR c:\backup
```

After running this batch file, if MKDIR fails, the result code is 1. However, a result code of 1 for MKDIR could mean multiple things. MKDIR sends an error message to STDERR. This error message is not logged because only STDOUT is . However, STDERR can be passed to STDOUT using redirection operators in the batch file. (For more information on STDOUT and STDERR, see Using command redirection operators on the MSDN® web site.)

```
1. @ECHO ON
2. ::This batch file displays the run keys from the registry
3. MKDIR c:\backup 2>&1
```

By simply adding the redirection operator marked in red, any messages passed to STDERR are also passed to STDOUT. This can be beneficial in determining the reason a specific line in a batch file fails.

Avoiding Common Mistakes

Common mistakes happen often and can sometimes be difficult to troubleshoot. Below is a list of mistakes to be aware of.

Use the Full Path

When there is doubt to whether the command is in the current working directory or in PATH, use the full path.

Never Use PAUSE

The PAUSE command is commonly used when manually testing batch files. This command should not be used when pushing a batch file with LANDesk because it stops a batch file from completing and prompts a user to press any key to continue. Since batch files are always silent when deployed by LANDesk, there is no way to answer this prompt and the batch file process remains alive until the computer is rebooted or until the cmd.exe process is manually killed. The Scheduled Task remains in an Active status until the timeout occurs.

Any attempts to deploy another batch file while one is already hanging at the pause command results in an error message such as:

```
Failed: Another installation is currently in progress
```

Answer All Prompts

Like PAUSE, other commands can prompt the user. When scripting with batch files, a prompts prevents a batch file from ever completing. The scheduled task will remain active until for hours until it times out. For example, deleting all files in a directory with the following command…

```
C:\> del c:\temp\*.*
```

…results in the following prompt.

```
C:\temp1\*, Are you sure (Y/N)?
```

When deployed through LANDesk, this prompt cannot be answered as batch files are always silent. This prompt causes the task to stay active until the task times out. A command line parameter can be used remove the prompt.

```
C:\> del c:\temp\*.* /Q
```

Also note that if a network drive is in use, trying to delete the drive results in a prompt. Adding a /y eliminates the prompt.

```
C:\> net use x: /del /y
```

Check Each Line for Spelling Errors

Spelling errors are very common and can cause detrimental results. Make sure each line is tested, and any spelling errors are avoided before deploy a batch file out with LANDesk.

Avoid Ambiguous Statements and Use Parenthesis When Necessary

When writing batch files, the intent may be to do two separate command but they may run together.

For example, look at the following IF statement that it used to handle a non-zero success code of 1.

```
IF %errorlevel%=="1" cmd /c ELSE GOTO end
```

At first glance, this appears to be a valid command. If errorlevel equals 1, then cmd /c should run and set errorlevel to 0. However, that is not what is happening. Because cmd /c normally expects a parameter, "ELSE GOTO end" becomes a parameter of cmd /c. So if errorlevel equals 1, then "cmd /c ELSE GOTO end" is run as a single command and fails.

This can be fixed with parenthesis. Parenthesis separates the conditional command from the ELSE function.

```
IF %errorlevel%=="1" (cmd /c) ELSE GOTO end
```

Sample Batch Files

This section contains sample batch files. These are not fully developed batch files and are only samples. Some batch files below require modification to even work. This content is for demonstration purposes only and LANDesk is not responsible for issues related to the use of these samples.

Sample 1 – Installing an MSI

```
REM
REM Copyright 2005 LANDesk(R) Software
REM
REM This batch file is used to demonstrate deploying an MSI package
REM for a network share that requires authentication.  Place batch file
REM in an anonymous access web share.
REM
REM TODO #1: SET core=<corename>
SET core=<corename>

REM TODO #2: Put authentication command here
NET USE \\%core%\share /USER:Domain\user password

msiexec /qn /I \\%core%\share\folder\install.msi ALLUSERS=1
REM The unauthenticate command changes ERRORLEVEL.
REM Preserve ERRORLEVEL with the following line.
SET msiresult=%ERRORLEVEL%

REM TODO #3: Put authentication disconnect command here
NET USE /del \\%core%\share /y
EXIT /B %msiresult%
```

Sample 2 – Sending a Message to the Core Server

```
REM
REM Copyright 2005 LANDesk(R) Software
REM
REM This batch file sends the first command line option back to the
REM Core Server in custom message.
REM

sdclient.exe /msg=%1
```

Sample 3 – Checking for the Existence of a File

```
REM
REM Copyright 2005 LANDesk(R) Software
REM
REM This batch file check for the existence of a file
REM
IF NOT EXIST "c:\file\to\find.txt" GOTO filenotfound
sdclient.exe /msg="The file was successfully found."
GOTO getout

:filenotfound
```

```
sdclient.exe /msg="The file was not found."

:getout
EXIT /B 0
```

Sample 4 – Adding a Local Scheduler Task

```
REM
REM Copyright 2005 LANDesk(R) Software
REM
REM This batch file is used to schedule the miniscanner to run
REM every time the IP address changes – 8.6 and later
REM

LocalSch.exe /exe="%LDMS_LOCAL_DIR:~0,-5%\miniscan.exe" /taskid=2001  /ipaddr
/freq=1
IF ERRORLEVEL==2001 EXIT /B 0\
```

Sample 5 – Running a VBScript

```
REM
REM Copyright 2005 LANDesk(R) Software
REM
REM This batch file runs a VB script file
REM Be aware that when the batch file is running as Local System
REM the vbscript must work under the Local System account as well.
REM Some vbscript functions must be run as a user and cannot work
REM when run as Local System.  If is the vbscript developers
REM responsibility to troubleshoot the vbscript.

cscript.exe //B vbversion.vbs
```

Below is vbversion.vbs. This file should be included in the Distribution Package as an additional file.

```
' vbversion.vbs
' Copyright 2005 LANDesk(R) Software
'
' Visual basic application that sends the version of the WScript
' as a custom message to the core server
 set WshShell = WScript.CreateObject("WScript.Shell")
 WshShell.Exec("sdclient.exe /msg=" & WScript.Version & "." &
WScript.BuildVersion)
```

Sample 6 – Giving Users Full Permission to a Folder

```
REM
REM Copyright 2005 LANDesk(R) Software
REM
REM This batch file gives the users group full rights to somefolder
REM

cacls c:\somefolder /E /G BUILTIN\users:F
```

Sample 7 –Local User Account – Adding an Account

```
REM
REM Copyright 2005 LANDesk(R) Software
REM
REM This batch file changes a local accounts password
REM
REM

net user NewAdmin password /add /comment:”This is my companies admin”
```

Sample 8 – Local User Account - Changing Password

```
REM
REM Copyright 2005 LANDesk(R) Software
REM
REM This batch file changes a local user account's password.
REM
REM The password as the first batch file parameter.
REM
REM WARNING: Leave echo off.  Turn echo on for logging purposes only.
REM Turning echo on will result in the password existing in a log file
REM on the client in clear text.

NET USER username %1
```

Note: This batch file is not needed for LDMS versions 8.61 and later as changing the local account passwords can be scheduled from the Core Server but is included for use with previous releases.

Sample 9 – Installing an SWD Package

```
REM
REM Copyright 2005 LANDesk(R) Software
REM

SWDpackage.exe /Ai /Ao /An /Ah-
```