Using DOORS to Update and Configure DOORS Clients

Kevin Murphy
Baselines Incorporated
kevin.murphy@baselinesinc.com

Prepared for the Telelogic Americas Innovation 2007 User Group Conference

Abstract

Telelogic DOORS is a tool used by many companies to manage their requirements. While the DOORS software is very powerful, DOORS administrators cannot centrally control a user's individual software configuration. Configuration in DOORS is done on a per-client basis, controlled by registry settings and icons.

What can a DOORS administrator do when the configuration has to change for users across the enterprise and IT resources are thin? This paper will discuss various methods a DOORS administrator can employ, focusing on using DOORS itself to facilitate configuration updates, lessening administrators' reliance upon their IT divisions.

This paper will focus on Microsoft Windows platforms, but many of the concepts herein can be used by administrators of UNIX DOORS clients to achieve the same goal.

Biography

Kevin Murphy has been using DOORS throughout most of his professional career. He has worked for BellSouth, BAE Systems, and ASCAP as a DOORS administrator, and has written DXL code used at Boeing, SAIC, and General Dynamics Land Systems on the Future Combat Systems program. Kevin has a BA in Journalism from the University of Memphis in Memphis, TN, where he graduated magna cum laude. Kevin recently started a consulting firm, Baselines Incorporated. For more information about Baselines Incorporated, visit http://www.baselinesinc.com.

The Problem

I was assigned a position as a DOORS Administrator on a project that was already underway and had DOORS rolled out to quite a few users. Due to politics, management decisions and the nature of work in general, the DOORS configuration at this company was not documented well and was quite customized.

It was also not a good configuration, set up in a way that led to DOORS crashing on many simple operations, such as brining up the find dialog box, to more serious problems, such as modules disappearing.

To connect to DOORS, the Data Security group had the DOORS admin enforce a policy: a user had to have access to a share on the network drive in order to start DOORS. To get access to this directory on the network, Data Security had to put the user in a special DOORS group. The purpose for this requirement was that even if a DOORS administrator bypassed process and just added a user to the DOORS database without involving Data Security, the newly added user could not connect to DOORS because they could not see the network share.

To meet Data Security's requirement, the DOORS administrators before me configured DOORS so that the *Home* value in the registry pointed at the network share. According to the DOORS manual, *Home* defines the folder containing the bin/ folder. The administrator who configured DOORS copied the contents of C:\Program Files\Telelogic\DOORS 7.1 to the network and set *Home* to point to it for everybody. Thus, even though doors.exe was running from the user's hard drive, almost everything else the user did ran off the network. Therefore, if a user didn't have access to the network share to where *Home* was pointing, DOORS would give a cryptic DXL error and exit.

This caused all sorts of issues, and sometimes buggy database behavior. A few unintended consequences:

- The DOORS help menu didn't work, due to security issues in Windows from running compiled HTML help files from a network share. Once Microsoft released a patch to fix the security issues, help files no longer worked within DOORS.
- Users patched their DOORS clients, but since *Home* was pointing to a network share, the patched DOORS.exe was not finding the exact versions of the files it expected under *Home*, as the patch updated C:\Program Files\Telelogic\DOORS and not the network drive. This caused simple operations, like Find and Go To, to crash the DOORS client. If Find can crash DOORS, how can the integrity of the database be trusted?
- The patching of DOORS was not a coordinated, planned effort, so users did not all run the same version of DOORS 7.1.
- Not everyone had *Home* set to the network share, so Data Security's requirement was not met by all clients.

- Modules would disappear unexpectedly.
- If the network hiccupped, even for a split second during a critical operation, the client is vulnerable, and data corruption is more likely to occur.
- Users connected to another company's database using this configuration, so these problems could affect the other company and also cause performance issues.
- There was no truly standard project-wide configuration that was enforced, and no guidelines on how to configure DOORS for new projects and databases, due to the nature of how quickly this configuration was put together.

As a conscientious administrator, I knew I had to fix the configuration, and as quickly as possible. But there were constraints to deal with.

- Over 100 DOORS clients to update in different buildings, sometimes located more than a mile apart
- Limited IT budget/support
- No dedicated DOORS admin had the time to visit each user's machine
- Custom DOORS installations (D:\ drive, Citrix, Doors 7 directory, Doors 7.1 directory, and maybe even other unknowns)
- Legal Department required a pop-up notice on login
- Data Security required a check for the network share
- Home needed to be configured for all users to run from their local DOORS installation
- DOORS users cannot be down for a prolonged period of time

Even with these constraints, there were two positive things about our environment.

- No UNIX machines ran DOORS clients, so I only had to focus on updating Windows clients
- Almost everyone running DOORS had administrator rights on their local machine

Many corporate Windows environments do not allow their users to have admin privileges. My company's environment did allow their users to have admin rights, thus the users could all install software and even configure their registries themselves, if desired.[1]

Before describing my solution to this problem, it is important to go over some basics of how the Microsoft Windows OS stores software configuration options and launches applications.

[1] While this fact made controlling the update much easier for me, it still is possible to update users' registries in many cases by putting configuration data in HKCU instead of HKLM.

The Registry, Environment, and Command Line

The Registry

When installing a DOORS client on a machine running Microsoft Windows, the install program asks for some information, such as the location of the DOORS server, and which machine runs the license server.

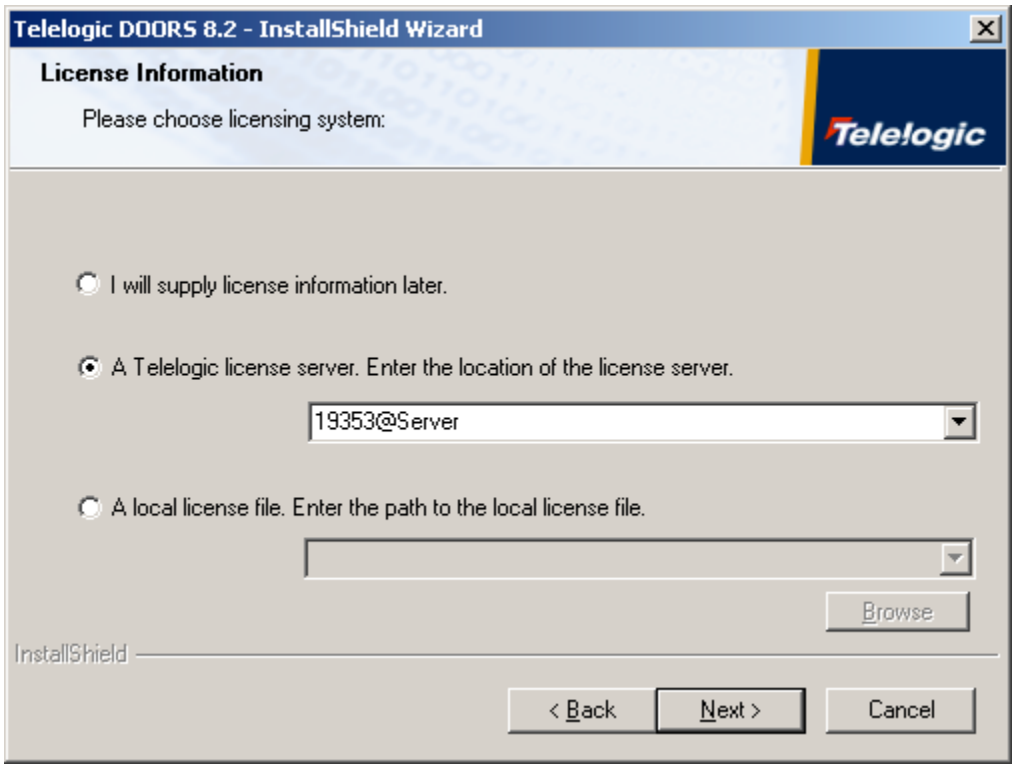

Figure 1: Defining the license server during client install

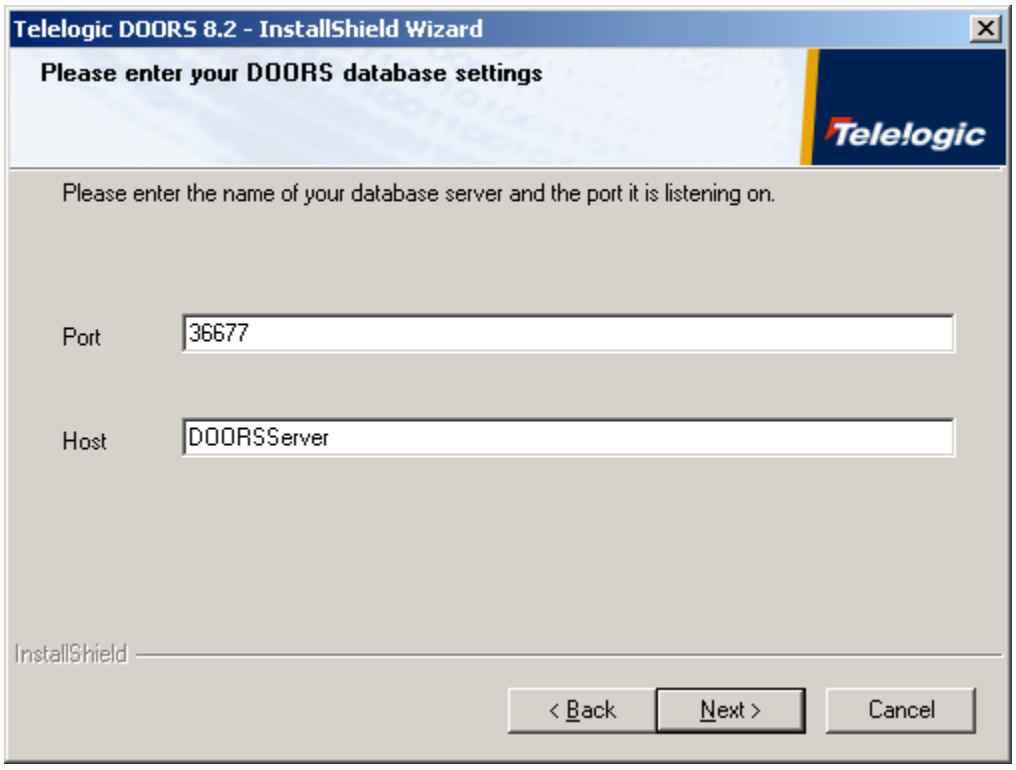

Figure 2: Defining the DOORS Server during client install

These values are stored in the Windows registry. In Windows, the registry stores configuration information for the software hardware installed on a system, as well as software configuration details for each user.

To view the registry, click **Start>Run**, and type regedit and click OK. The Registry Editor appears.

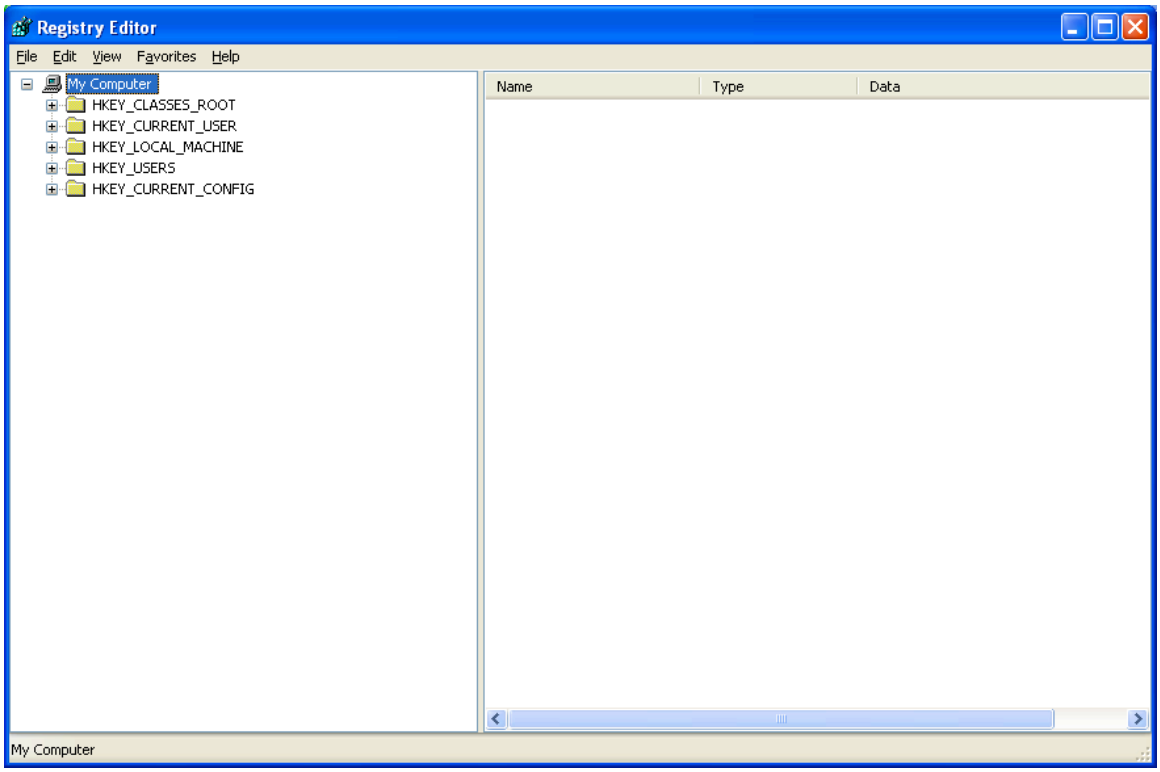

Figure 3: The Windows Registry Editor

CAUTION: Someone who does not know what they are doing should not change any registry values. It is relatively easy to mess up a registry and your system may not be able to recover from it. BE CAREFUL.

To find DOORS entries choose **Edit>Find**, and search for Telelogic.

The AppID key, or maybe some other key may be returned. If this is the case, press F3, to repeat the find. There may be many entries with the term "Telelogic."

Software configuration in the registry can be found in two places: HKEY_LOCAL_MACHINE (HKLM) and HKEY_CURRENT_USER (HKCU).

Telelogic software information is found in HKLM\SOFTWARE\Telelogic. From there, it is easy to see where DOORS configuration information is stored.

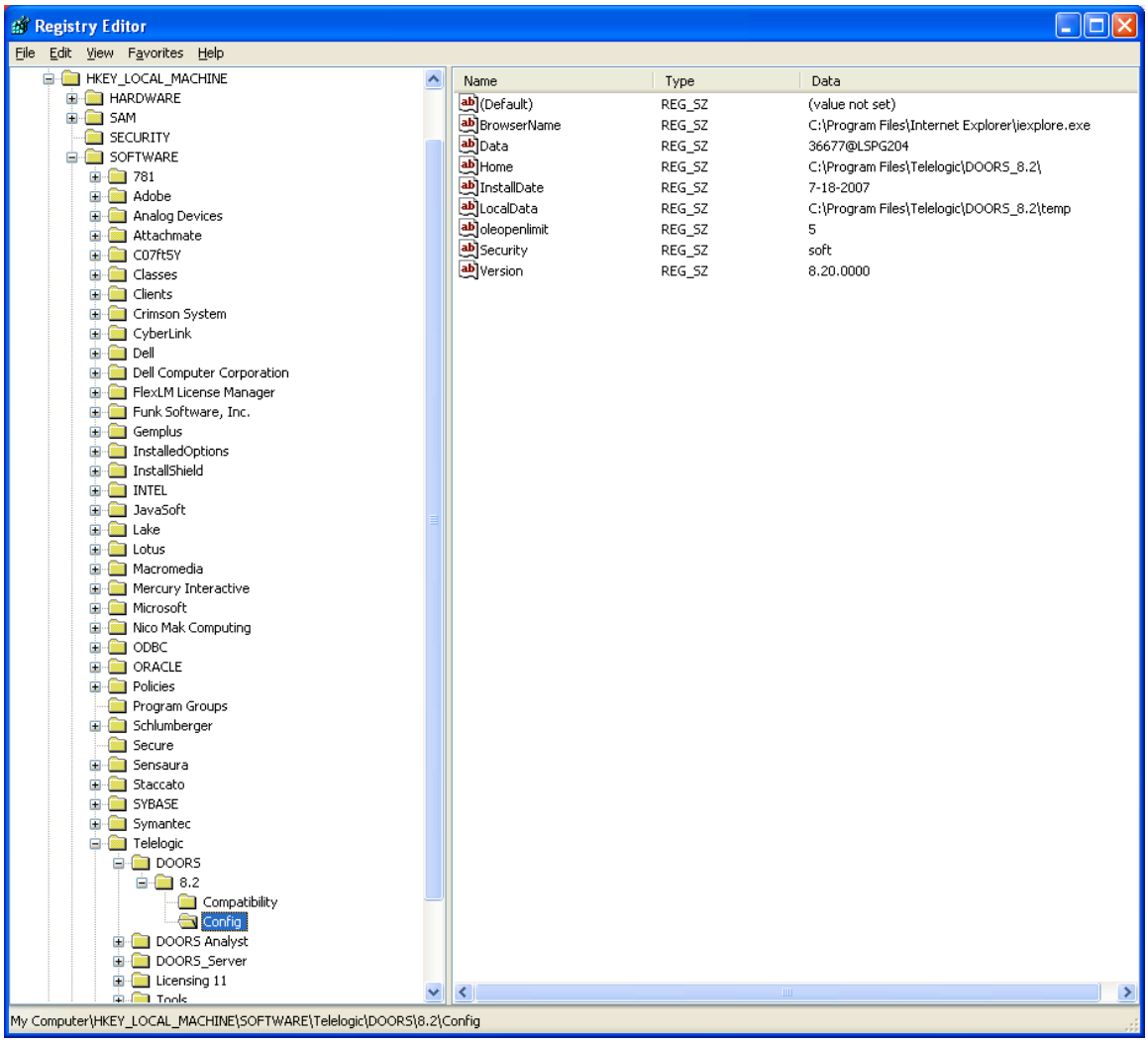

Figure 4: HKLM/SOFTWARE/Telelogic/DOORS/<VERSION>/Config

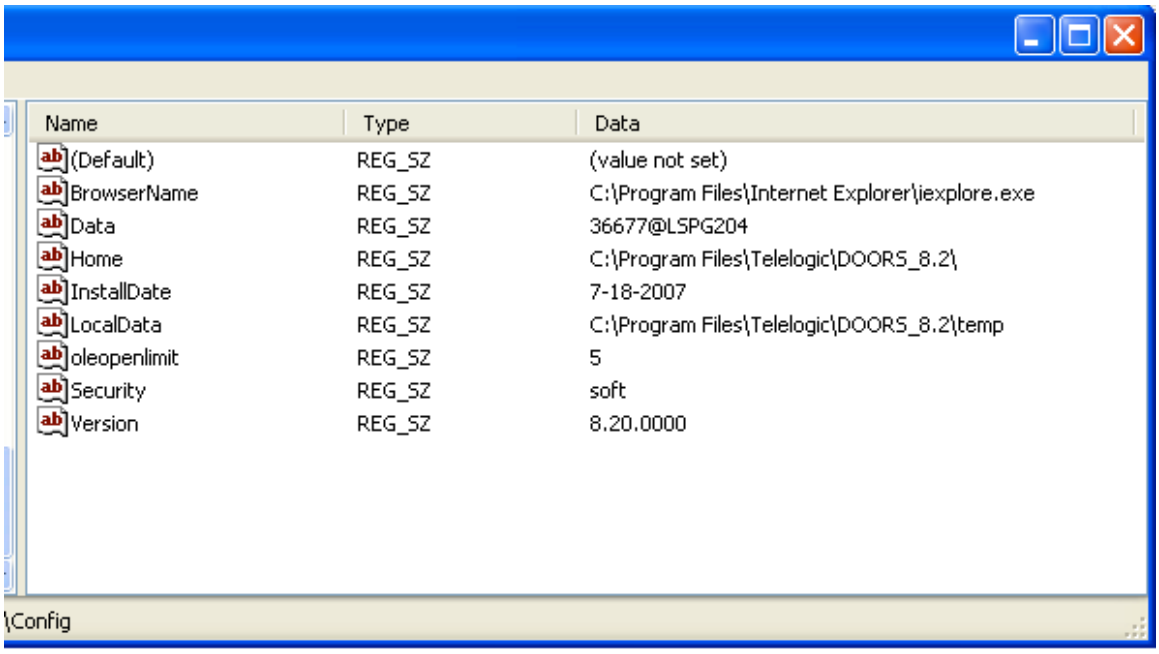

Figure 5: Values and Data for the DOORS Config key

This path, HKLM/SOFTWARE/Telelogic/DOORS/<VERSION>/Config, is called a *key*. Keys are represented by folder icons.

In this key, it is easy to see what options are set and what the options mean. BrowserName is the path to the Web browser DOORS will use when clicking on a URL. According to Figure 5, the Web browser that DOORS will use is Internet Explorer.

Each option is called a ***Value***. Each value contains ***Data***. The Data for BrowserName is "C:\Program Files\Internet Explorer\iexplore.exe."

There are other values: Home, Data, oleopenlimit, and more. Changing the data for these values will change the default behavior of DOORS.

These values all apply to the HKLM Software key. Navigating to the HKCU Software key shows different values.

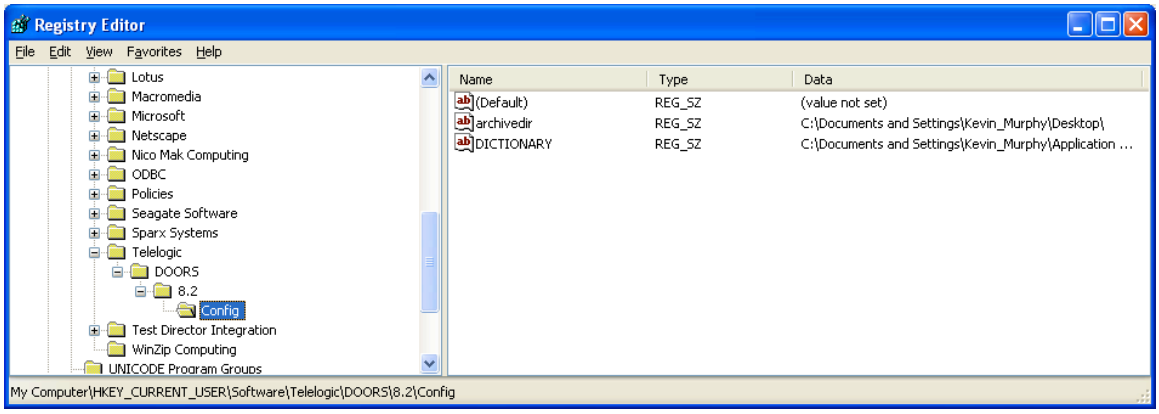

Figure 6: /HKCU/SOFTWARE/Telelogic/DOORS/<VERSION>/Config

Here, the archivedir and DICTIONARY values are stored.

If the same values are stored in both HKLM and HKCU, the HKCU value overrides the HKLM value.

Changes made to the registry are instant. There is no saving of the registry; once a change is made, it is made.

The Environment

In Windows, an environment variable may be defined by Administrators, users, and even some software install routines. Whereas the registry controls configuration of almost every piece of software on the machine, an environment variable may also be defined to override or complement registry settings, if desired.

To see the Environment Variables dialog window, right-click My Computer, choose Properties, then the **Advanced** tab, and finally click the **Environment Variables** button.

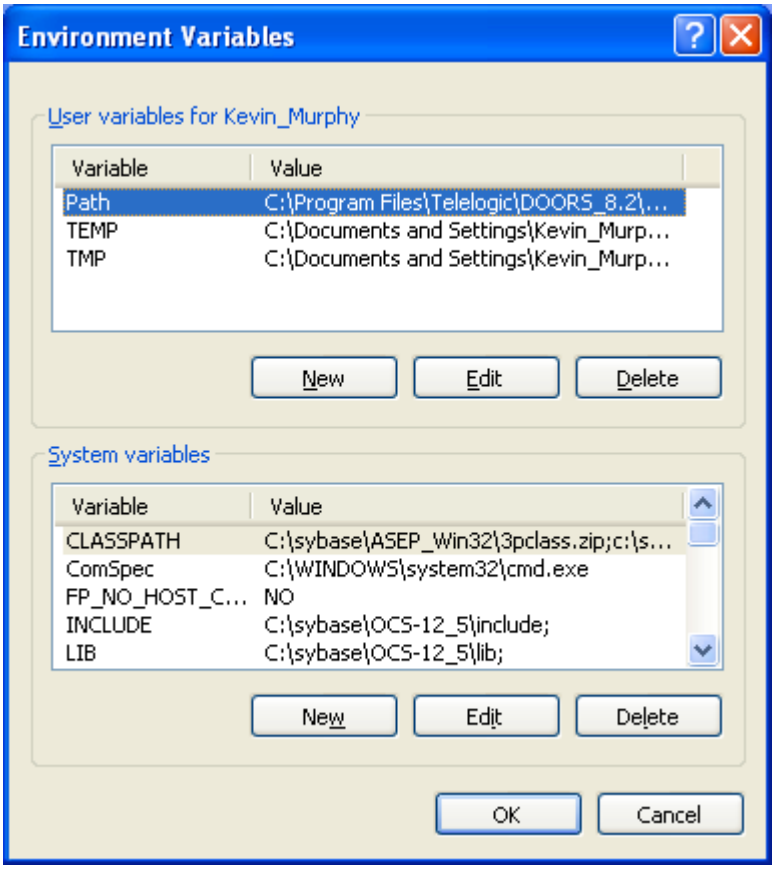

Figure 7: Environment Variables Dialog

The top half of the Window consists of *User Variables*, while the bottom half of the Window consists of *System Variables*. But both do the same thing as the registry—they store file, path, and configuration information.

Environment variables consist of a *Variable*, and a *Value.* This is exactly the same as the Registry's Value and Data fields.

As with the registry, the user variable overrides the system variable when the same variable name is used.

What is the point of having environment variables in Windows if the registry can do the same thing? Environment variables can be accessed more readily by users than Windows registry entries.

To use an environment variable on a command line, it is necessary to surround the variable name by % symbols. Using the system environment variables listed in Figure 7, typing the following on the command line:

```
echo %LIB%
```

would return c:\sybase\OCS-12_5\lib;

An environment variable, therefore, is merely just a useful shortcut.

The Command Prompt

When a program is run from the Start Menu, what is really happening most of the time is that the program is being executed with a command and maybe some options, called *switches*. Any switches not included in the command may be read from the registry.

Consider DOORS. Upon going to **Start>Programs>Telelogic>Telelogic DOORS** and clicking the DOORS icon, doors.exe is being run from wherever it was installed, using the data from values in the registry.

There are many switches that can be used with doors.exe. These switches are covered in the DOORS Help manual under the topic "Summary of command line switches."

To get to the command line in Windows, navigate to **Start>Programs>Accessories>Command Prompt**. Alternatively, **Start>Run**, typing "cmd" and clicking **OK** does the same thing.

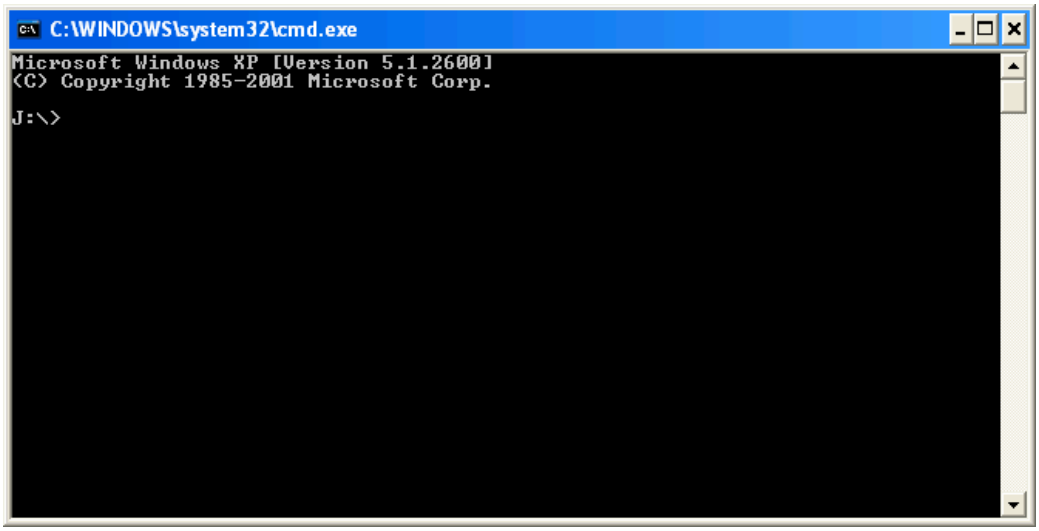

Figure 8: The Windows command prompt

To start DOORS using a switch or multiple switches, one must first navigate to where DOORS is installed. For the purpose of this paper, it will be assumed that DOORS is installed in its standard location on the C: drive.[2]

Typing "C:" without the quotes, and pressing enter, ensures the command line brings you to your C: drive. From here, type the following, pressing enter after each line.

```
cd Program Files
cd Telelogic
cd DOORS_8.2[3]
cd bin
```

Typing dir and pressing enter will return a list of files. One of those files should be doors.exe.

From here, to start DOORS with a switch, simply type

```
doors -o READ_ONLY
```

DOORS will launch, but now double-clicking a module will open the module read-only by default.

It's also possible to connect to other DOORS databases via a command line.

doors –d port@database

[2] If DOORS was not installed on the C: drive, just check the registry settings in HKLM to determine where DOORS was installed.

[3] This step may be different depending on your DOORS version. Typing "dir" and pressing enter will show you a list of files and directories.

So…

doors –d 36680@192.168.1.30 –o READ_ONLY

…would open a DOORS client, point it at the database on computer 192.168.1.30, connecting to port 36680, and setting the configuration so that double clicking formal modules opens them in read only mode.

A Windows shortcut is really nothing more than a saved command with switches. To take the above command and create a shortcut, perform the following steps.

1. Right-click the Windows desktop and choose **New>Shortcut**.

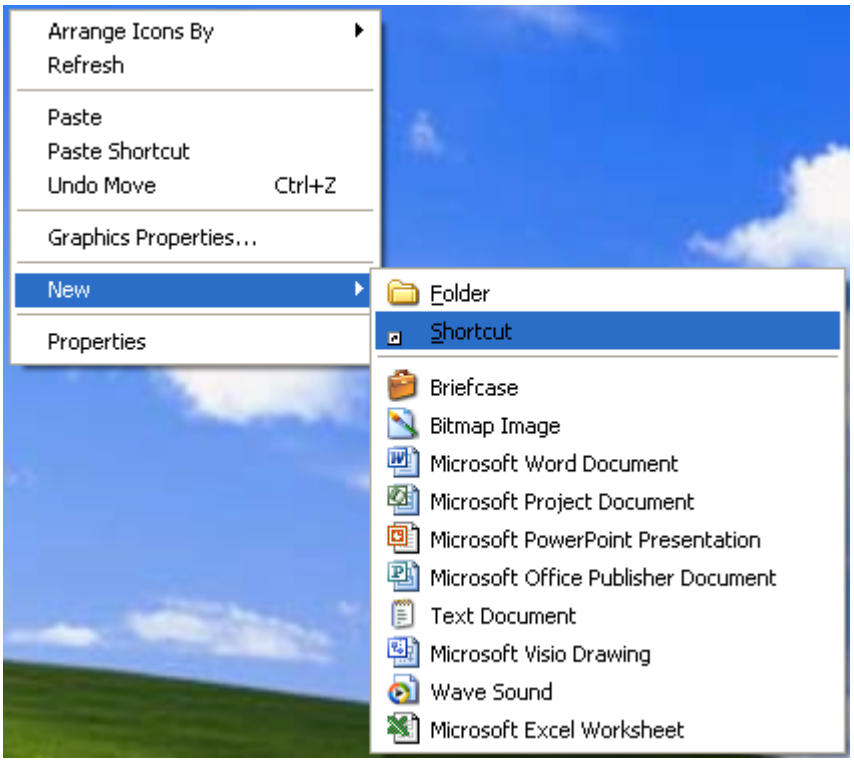

Figure 9: Creating a shortcut on the desktop

2. The Create Shortcut dialog appears.

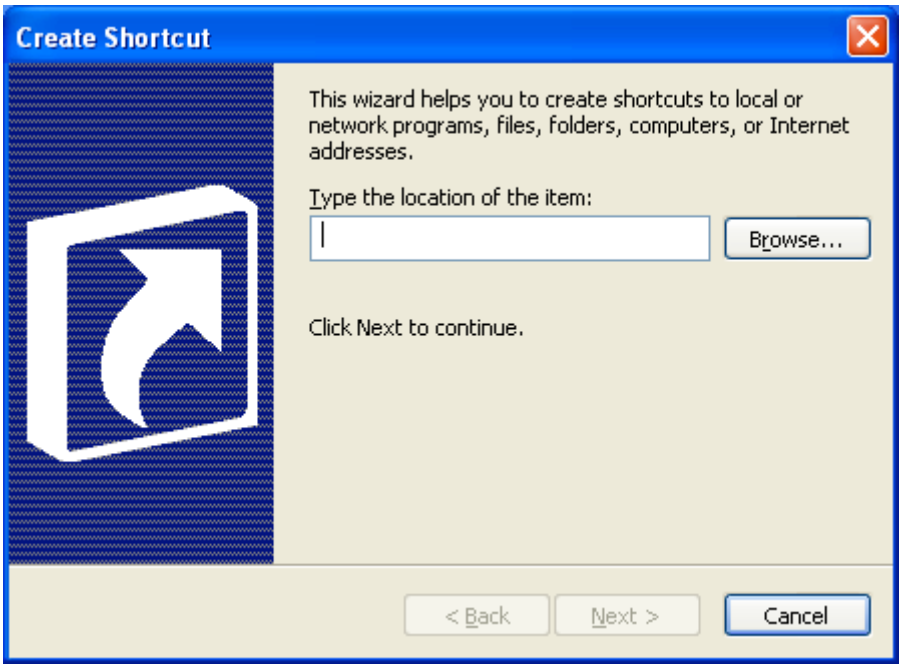

Figure 10: The Create Shortcut dialog

3. Click **Browse** and navigate to Program Files\Telelogic\DOORS_VERSION\bin\ and choose doors.exe.
4. After the doors.exe", type
   `-d 36680@192.168.1.30 -o READ_ONLY`
5. Click Next.
6. Name the shortcut "Other DOORS Database."

Chances are the database will not be found when DOORS is launched from this shortcut. That is because there is likely no database running at 192.168.1.30 on port 36680 on the network. However, going back and launching DOORS from the Start menu, or whichever way the reader is used to launching DOORS will give the normal, expected results.

One limitation of Windows shortcuts (in XP and below) is that they can only be 255 characters long, and there are quite a few options to store for the switches. This is where knowledge of environment variables comes in handy, as a savvy DOORS admin could define environment variables that housed over 255 characters, and then use those environment variables in a DOORS command tied to an icon.

In other words, if an environment variable named OTHERDB was defined with a value of "36680@192.168.1.30" then the following statement:

doors –d 36680@192.168.1.30 -o READ_ONLY

is identical to

doors –d %OTHERDB% -o READ_ONLY

and although the two statements are equivalent, the first statement contains 42 characters, while the latter statement contains 31 characters. Further still, defining OTHERDB as "-d 36680@192.168.1.30 –o READ_ONLY –O READ_ONLY", and removing the –o switch now gives us the command:

doors %OTHERDB%

This command is 16 characters long, which is less than half the length of the original statement, with more configuration options set than the original command!

Keep in mind that command line switches override registry settings. If –d is defined within a command line command, then the parameter for the Data value in the registry is ignored. In other words, without the –d switch, DOORS looks in the registry to determine the host database. If the –d switch is present, DOORS ignores the registry entry that defines which database to connect to.

Given this knowledge of registry settings, environment variables, and the command line, it is possible to use these tools to create and update a standard configuration for DOORS across the enterprise, and to use DOORS to facilitate the implementation.

Proposing a Solution

I came up with a proposed solution, but before implementing it, I met with our IT group so they knew exactly what I was going to do.

As DOORS Administrators, we are usually not part of IT, yet we do run a database. This is a unique position to be in, as most IT divisions do not want to be responsible for day-to-day DOORS activities, rather they care more about data backups and server maintenance, by and large.

While you may be trying to update your clients, you must keep in mind that by updating DOORS software without having IT do it, that they are going to have to support anything you may break.

If you have a thought-out plan before meeting with your IT group, then they will likely appreciate you saving them time and effort. Further, they can inform you of potential issues with your approach.

In my case, if the update did not work on some users' machines, IT was prepared to do reinstalls on those machines. They'd much rather reinstall DOORS 10 times than over 100 times.

I created a flow chart and explained to IT exactly which registry keys and values this update would touch, as well as which DOORS files would be copied and overwritten. I also gave them all of my supporting files. We worked together to update their documentation on how to install and configure DOORS for new users, so that every new user from then on would have the correct configuration by default. Further, in the course of meeting with IT, I was informed that users could not update their registry settings if they were running Citrix clients, therefore I had to rely on IT to configure the DOORS icons and registry settings for Citrix users, and let those users know to ignore the update.

This was also a good time to review with Data Security the exact purpose for the network drive, and to also contact Legal to see if our sign-on notice needed updating. While this did add time to actually implementing the update, it saved time overall because I may have had to create another update to fix an important issue that I originally overlooked.

I cannot emphasize this enough: DOORS is a public database with many stakeholders at most companies. It is not just *your* database. It is shared. Every user is potentially affected by what you do. Let as many people know what your specific plans are as possible. You don't want to make any enemies in other groups, or your own, due to your ill-applied good intentions.

The Solution

With so many different constraints, the only thing I knew with certainty was that every one of my DOORS users could connect to DOORS. Not everyone had access to the network share. Not everyone had *Home* configured in the registry. Not everyone had the same icons. But everyone had access to the same DOORS database.

I created a project called "Update DOORS" and placed a module within that project. When opened, this module runs a trigger that performs the update.

In other words, now any DOORS user can update their DOORS client configuration by opening a module in the database.

The trigger consists of DXL which sets the registry settings appropriately. Then the DXL executes a DOS batch file and a VBScript file, at which time the user gets a message that the update has completed and DOORS will exit. The reason for exiting DOORS is to ensure that the user restarts DOORS with the new configuration.

Using batch files allowed me to copy files more reliably than DXL's copyFile command. The VBScript allowed me to create DOORS icons in the user's start menu and desktop, and just as importantly, VBScript allowed me to delete the old icons it found.

I had originally put the text that appears in the legal pop-up in a new startup.dxl file that was to be created within the update, but I thought better of it and put it on the network share, in case of future updates from Legal.[4] I had startup.dxl point to an include file on the network, and the include file housed the pop-up function and text.

A great benefit of this approach is now I have a file on the network that I know is touched at every log in. I can add other functions to the include file that can update clients' software. Like triggers, this can be a great thing, yet at the same time, a dangerous thing. The benefit, though, is that I now have another option in which to update the DOORS configuration throughout the enterprise.

Motivating the Users

Some users will do whatever an admin tells them. Others will not. Windows administrators can roll out patches over the network in the background. DOORS administrators have no such luxury. There was no way that I could force any user to install my update. But I needed them to. The integrity of the database, and another companies' database, was at stake.

How did I solve this problem?

I sent out an email informing everyone of the new company standard configuration for DOORS. I explained that as a user, they could update DOORS themselves by going to the "Update DOORS" module.

[4] Write access to the network share is given to very few people, so this was a relatively safe way to do this.

I told them that if they didn't do this within two weeks, their DOORS account would be disabled.

Almost everybody updated DOORS. A tiny little threat went a long way.

Update Problems

As expected, not everyone was able to successfully apply the update, due to the many different DOORS installs. The update appeared to be successful, but upon restarting DOORS, DXL errors prevented users from entering the database. However, in every single case, the fix was simply to uninstall and reinstall DOORS, and then apply the update I wrote. Only about 10 to 15 percent of the updates did not work.

Conclusion

In many cases, updating an errant DOORS configuration can be done completely by a DOORS administrator versed in DXL and Windows. The benefits of an administrator using DOORS to update DOORS may be:

- Lower IT cost
- Faster time to deployment
- More control over the client configuration (it gets done the same way, every single time)
- More control over the deployment and configuration
- Because of the required network check, it is easier to roll out configuration updates in the future
- Working with other groups, the DOORS administrator is likely forced to document exactly what the update does, in great detail
- Less reliance on schedules of other groups
- If a user can already access DOORS, they can access the update

Appendix A: Useful DXL snippets for updating DOORS

All of the code below was written for DOORS 7.1 running on Windows XP Professional. Not all features and paths may be the same for different versions of Windows and DOORS.

Snippet 1: Registry Constants (DOORS 7.1, Windows XP Professional)

```
const string WindowsRegPath = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion"

const string SystemRoot = "SystemRoot" //Windows Registry Key for Windows Install path

const string DOORSConfigRegPath = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Telelogic\\DOORS\\7.1\\Config"

const string DOORSHomeKey = "Home" //Windows Registry Key for DOORS Home switch

const string DOORSRegKey = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Telelogic\\DOORS\\7.1"

const string InstallationDirectory = "InstallationDirectory"  //Windows Registry Key for DOORS
     //install path. DOORS HOME should be this plus \\bin

const string DOORSExecutable = "doors.exe"
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

Snippet 2: Get Windows Install Path

```
string getWindowsInstallPath() {
      //Returns path where Windows is installed
      //DXL file must include the constants in Snippet 1 for this function to work.
      return getRegistry(WindowsRegPath, SystemRoot)
}
```

Snippet 3: Escape Slashes

This function will make registry entries of paths usable in DXL. For example, it will take a Buffer of "C:\Windows" and return "C:\\Windows".

```
Buffer escapeSlashes( Buffer source ){ //This function changes all \ to \\ within source parameter.
      Regexp slash = regexp "\\\\"
      int from = 0
      int l = length(source)-1 //-1 because DOORS buffers start at pos 0, length function starts at 1
      string aftermatch = null
      Buffer result = create
      while ( search( slash, source, from ) )
      {
            int offset = end 0 // position of end of the match within source
            string match = source[from:from+offset] //get string to first match
            result += match "\\" //append a \ to result
            aftermatch = source[from+offset+1:l] //capture buffer after the match.
             from = from + offset + 1 //start next search after match
      }
//put rest of string into result
result += aftermatch
return result
}
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

```
//To use this function…
string strWinPath = getWindowsInstallPath
Buffer tb = create
tb = strWinPath
tb = escapeSlashes( tb )
strWinPath = stringOf tb
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

Snippet 4: Run any command line command

```
//Example 1
string strCmd = "ping www.telelogic.com"
system ( strCmd )

//Example 2
strCmd = "c:\\temp\\mybatchfile.bat"
system ( strCmd )
```

Snippet 5: Execute a VBScript

```
//Define VBScript location
String VBSCriptFile = "c:\\temp\\myVBScriptFile.vbs"

//Get windows install path
string strWinPath = getWindowsInstallPath //function defined in Example 2 of this paper

//format path for DXL use
Buffer tb = create
tb = strWinPath
tb = escapeSlashes( tb )
strWinPath = stringOf tb

//format entire command
String strExec = strWinPath "\\system32\\cscript.exe " VBScriptFile ""

//execute VBScript
system ( strExec )
```

Snippet 6: copyFile function[5]

```
//The first parameter is the source filename, the second parameter is the destination filename.
//Destination directory must already exist. In the example below, if C:\temp doesn't exist, no error
//is returned, but the file is not created.

copyFile("C:\\Windows\\clock.avi", "C:\\temp\\kcolc.avi")
```

[5] Sometimes the copyFile function may fail due to system policies and access rights. If this happens, creating a batch file that calls a copy command may work instead. Just create the batch file, and execute it in DXL via a system( ) call. In particular, I had problems copying files to directories beneath C:\Program Files.

Snippet 7: Does a directory/file exist?[6]

```
//*************************
bool fIsFile_Stat(string NameFile)
{     // Use file "Stat" to determine if the file exists.
      Stat s
      bool  IsFile = false
      s = create NameFile
      if (!null s)
      {  IsFile = true
         delete s
      }
      // print "fIsFile\t'" NameFile "'\t" IsFile "\n"
      return IsFile
}     // end fIsFile_Stat

//*********************
bool  fIsFile_Exist(string FileName)
{     // Use file Exists to determine if the file exists
  if (canOpenFile(FileName, false) or !null readFile(FileName))
```

[6] Code by Louie Landale, posted on Telelogic DOORS Forums at https://support.telelogic.com/en/doors/forums/messageview.cfm?catid=17&threadid=2903&STARTPAGE=1.

```
  { // print "File Exists: " FileName "\n"
    return(true)
  }else
  { // print "File does NOT exist: " FileName "\n"
    return(false)
  }
}     // end fIsFile_Exist()

//*************************
bool  fIsDir_Stat(string NameDir)
{     // Use Stat to determine if the name is a windows directory
      // Is the name a Windows Directory; use stat to figure it out.
      // Thus function fails to detect the case where a drive letter is valid
      //    yet there is nothing in the drive, such as no CD in drive "D:".
    Stat s
      bool  IsDir = false
    s = create NameDir
    if (!null s && directory s) IsDir =  true
    if (!null s) delete s
//    print "fIsDir1  '" NameDir "'  " IsDir "\n"
    return IsDir
}     // fIsDir_Stat()
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

```
//*************************
bool  fIsDir_Dir(string NameDir)
{     // Use directory to determine if the name is a windows directory,.
      // Determine whether the specified name is a directory whose contents
      //    are readable by the current user.
      // Thus, if there is no CD in the drive, "D:" should return false.
      string NameFile = ""

      bool       IsDir = false
      if (!null NameDir)
      {
            // Drive names respond to the File Exists check,
            //    wereas other names respond to the "directory" check.
            //    I don't know why.
//    int   NameLen = length(NameDir)
//    if (NameDir[NameLen-1] == ':')
//    {  print "Colon\n"
//       return (fIsFile NameDir "/")
//    }
//    else
                  // Does the specified name return a valid 'directory' reference
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

```
                //
        Directory__ dir = directory(NameDir)
        if (null dir)
        {  print "\tDirectory__ is null\t" NameDir "\n"
           IsDir = false
        }
        else
        {
           noError()
           for NameFile in directory(NameDir) do
           {  break
           }
           string ErrMess = lastError()
              // Its a directory if there was no DXL error.
           // if (!null ErrMess) print "\t" NameDir "\t" ErrMess
           IsDir = (null ErrMess)
        }
     }
// print "isDirectory '" NameDir "'\t" IsDir "\n"
     // bool    IsDir1 = fIsDir1(NameDir)
     // return(IsDir1)
     return(IsDir)
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

```
}     // fIsDirIDir()

//*********************
void  DoTest(string Name)
{
      bool  IsDirDir    = fIsDir_Dir(Name)
      bool   IsDirStat   = fIsDir_Stat(Name)
      bool  IsFileStat  = fIsFile_Stat(Name)
      bool  IsFileExist = fIsFile_Exist(Name)
      print Name "\t" IsDirDir "\t" IsDirStat "\t" IsFileStat "\t" IsFileExist "\n"
}     // end DoText()

//Example Usage
print "Name \tdDir\tdStat\tfStat\tfExist\n"

DoTest("C:")
DoTest("c: ")
DoTest("c:\\")
DoTest("d:")           // No CD in drive D
DoTest("d:\\")         // No CD in Drive D
DoTest("c:\\xxx")           // Doesn't exist
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

```
DoTest("c:\\t1.txt")   // size 0 bytes
DoTest("c:\\t2.txt")   // size 1 byte
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

Snippet 8: Exit DOORS

Putting the exit() function within an eval_ statement ensures that DOORS will quit.

```
eval_("exit_()")
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

Snippet 9: Implementing a Trigger

```
//the below should all be on ONE LINE, and you want to ensure that all users have read access
//to the DXL file in the #include statement.


Trigger t = trigger( "DOORSUpdate", module->"DOORS Update", post, open, 5, "#include
<N:/Doors/Path_To_DXL/updateDOORS.dxl>")
```

Appendix B: Alternative/Supplemental Methods of Updating DOORS Clients

Batch Files

Batch files are plain text files that consist of command line commands.

Create a text file. Open it for edit. Type the following.

```
mkdir c:\temp\clockfile

copy C:\Windows\clock.avi C:\temp\clockfile\kcolc.avi
```

Save the file as copyclock.bat. Double-clicking the file will execute the commands.

DXL can execute batch files with the system(string fullPathToBatchFile) command as demonstrated in Snippet 4 in Appendix A.

Batch files are useful in updating DOORS because it is easy to create directories and copy files.

VBScript

Visual Basic script, or VBScript files, are plain text files that run Visual Basic commands when executed. VBScripts are useful when paths are unknown and need to be determined dynamically, such as the path to a user's desktop. VBScripts can also be used in place of icons and environment variables to start DOORS in instances when the command to start DOORS is greater than 255 characters.

Deleting and Creating DOORS Shortcuts with VBScript

Create a new text file and save it as createDOORSIcons.vbs. Insert the text below.

```
dim DOORSPath


set objShell = WScript.CreateObject("WScript.Shell")


'Get the directory in which DOORS was installed.
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

```
DOORSPath = objShell.RegRead("HKEY_LOCAL_MACHINE\SOFTWARE\Telelogic\DOORS\7.1\InstallationDirectory")
DOORSPath = DOORSPath & "\bin\doors.exe"

'Create Desktop Icons
strDesktopFolder = objShell.SpecialFolders("Desktop")
Set objShortCut = objShell.CreateShortcut(strDesktopFolder & "\Example DOORS Icon.lnk")

'chr(34) is the quotation mark symbol, used for readability
objShortCut.TargetPath = chr(34) & DOORSPath & chr(34)
objShortCut.Arguments = " -d 36677@database -a N:\Doors\addins -L N:\Doors\addins " & _
"-A N:\Doors\addins -o READ_ONLY -O READ_ONLY -k"
objShortCut.Save

set objShortCut2 = objShell.CreateShortcut(strDesktopFolder & "\Another DOORS Example.lnk")
objShortCut2.TargetPath = chr(34) & DOORSPath & chr(34)
objShortCut2.Arguments = " -O READ_ONLY -o READ_ONLY -k"
objShortCut2.Save

'Create Start Menu Items
strAllStartMenu = objShell.SpecialFolders("AllUsersPrograms")
strTelelogicGroup = strAllStartMenu & "\Telelogic"
```

```
'Delete all preexisting DOORS icons
set fso = CreateObject("Scripting.FileSystemObject")
set mainfolder=fso.GetFolder(strTelelogicGroup)
set filecollection = mainfolder.Files

For Each file In filecollection
     if inStr(file.name, "DOORS") then
          if right(file.name, 3) = "lnk" then
               file.delete
          end if
     end if
Next

'Place icons into Programs Menu

set objShortCut2 = objShell.CreateShortcut(strDesktopFolder & "\Start Menu Example.lnk")
objShortCut2.TargetPath = chr(34) & DOORSPath & chr(34)
objShortCut2.Arguments = " -O READ_ONLY -o READ_ONLY -k"
objShortCut2.Save

set objShortCut = Nothing
set objShortCut2 = Nothing
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

```
set objShell = Nothing
WScript.Quit(0)
```

Setting Temporary Environment Variables and Launching DOORS with VBScript

The following VBScript code only sets the environment variable for the duration that the script is run. Once the command line is executed and the script has ended, the environment variable created no longer exists.

Create a new text file and save it as launchDOORS.vbs. Insert the code below.

```
On Error Resume Next


Dim objShell
set objShell = WScript.CreateObject("WScript.Shell")


set objEnv = objShell.Environment("PROCESS")
objEnv("ADDINS") = "C:\temp\addins;C:\temp2\addins;C:\temp3\addins"
objShell.Run """C:\Program Files\Telelogic\DOORS_8.2\bin\doors.exe"" -a " & objEnv("ADDINS")


set ObjShell = Nothing
WScript.Quit(0)
```


Author: Kevin Murphy (kevin.murphy@baselinesinc.com)

Reg Files

When a user double-clicks a .reg file, the registry is automatically updated as long as the user has rights to update that portion of the registry.

To create a .reg file, do the following.

1. Start Regedit by choosing **Start>Run** and type regedit. Click **OK**.
2. Navigate to the key containing the data to be updated.
3. Set the data to the values accordingly.
4. Click **File>Export**…

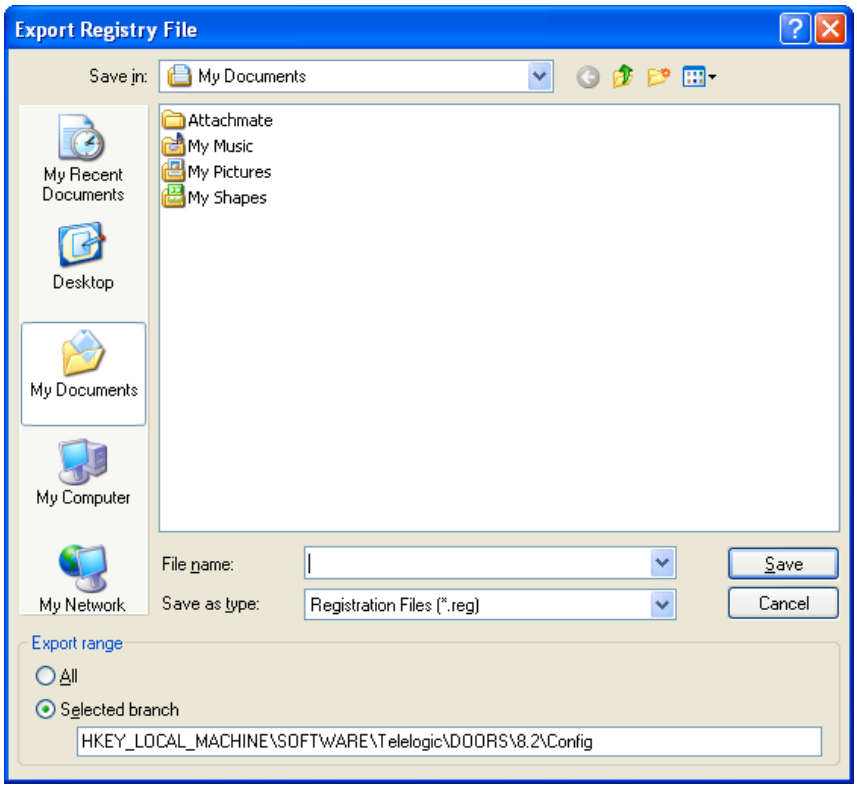

Figure 11: Export Registry File Dialog

1. Enter a file name
2. Ensure that the value for "Selected branch" is correct.
3. Click **Save**.

Appendix C: Sample Flow Given to IT Department

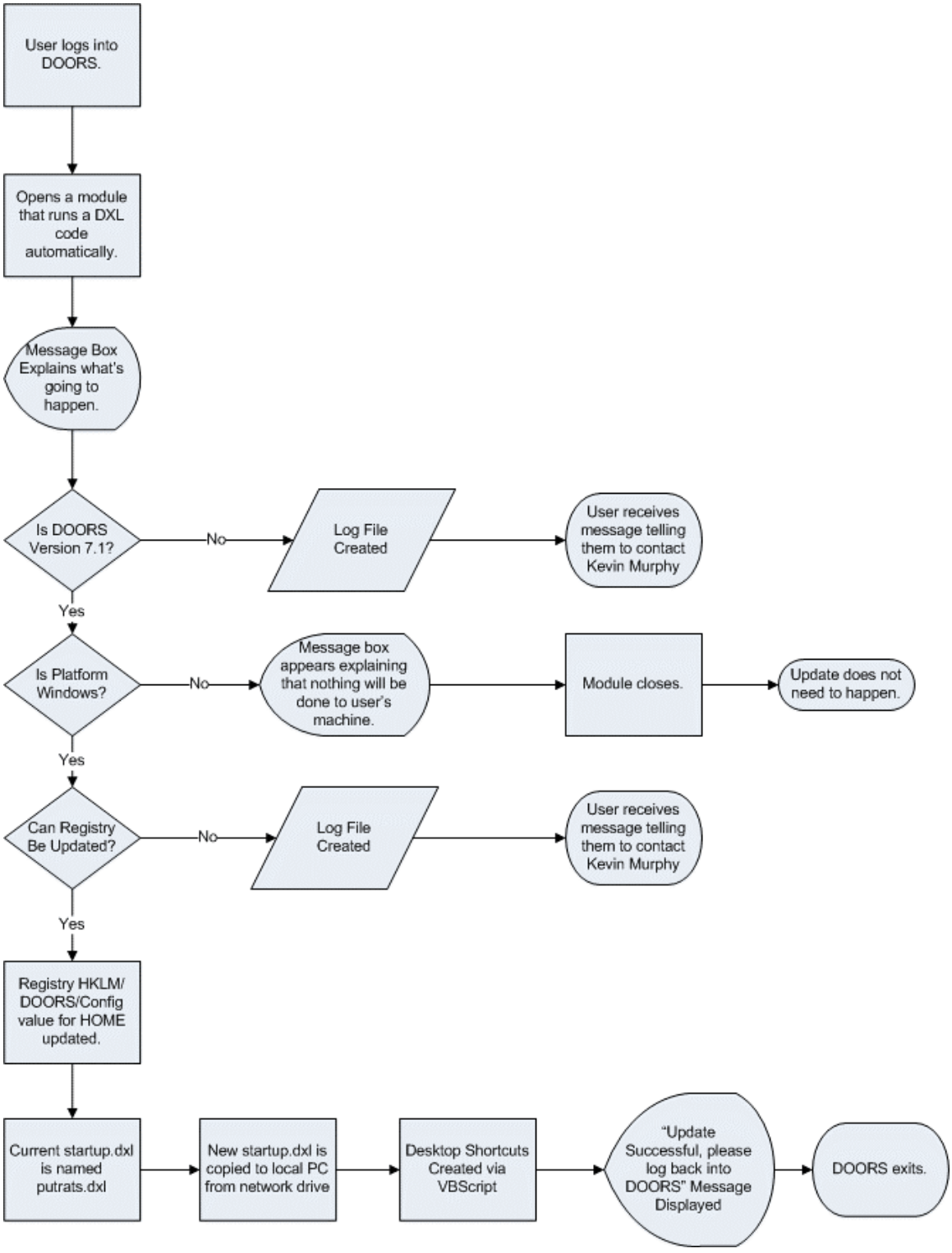

Figure 12: Flow chart detailing update procedure