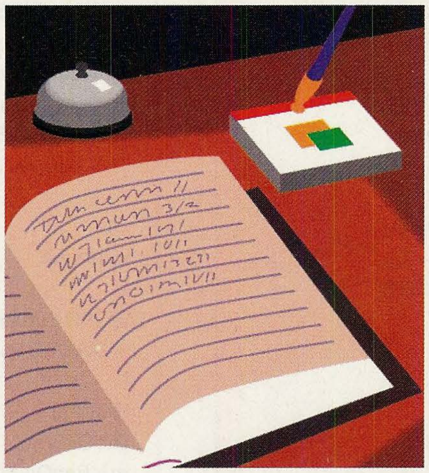
Douglas A. Hamilton

The Windows NT Registry



SCOTT BALDWIN

PERHAPS THE LEAST understood Windows NT feature is the registry. What the registry does is gather together all the information that the operating system needs to know about the hardware (the type of processor, addresses of the peripheral controllers and so on), the system software (where the swap files are located, what time zone the machine is in, recognized filename extensions and network protocols) and each user (his or her preferred screen colors, environment variables and Program Manager choices). The registry also serves as a collection point for per-application information that otherwise would be scattered through myriad .INI files.

The objectives of the registry are to:

• Overcome the confusion of so many different special files typically used to configure other systems

• Structure configuration information so that it can be separated by system, user and application

• Provide a secure interface controlling access to all of this information

• Allow configuration information to be shared across a network

• Support a standardized user interface via Control Panel

Several of these objectives arise from the view of Windows NT as a system designed for a networked environment with lots of machines and lots of users. The NT design team imagines a world in which you might sit down in front of any machine on the network, log in and have it act indistinguishably from your own "home" machine. If you're the owner of the machine, it shouldn't be like having someone else drive your car, and you find the seat's at the wrong height, the mirror's at the wrong angle and the radio's on the wrong station.

The result is a standardized database of information about your personal preferences that can be shared in a secure manner across a network. With this information in the registry, as soon as you log in, NT can go ask your own machine everything it needs to know about you. No matter where you log in, it's just like home. And because it's secure, all that customization is still kept private, accessible only to you even though it's available to you anywhere you go.

Internally, data is kept in the registry in a tree structure rather like the file system. To access any particular piece of information you specify a path, called a key, for traversing down the tree from one of four predefined starting points. Individual leaves of the tree, called values, typically contain a name (though some values can be unnamed) plus the binary or character string data associated with that name. The four predefined starting points are:

Figure 1. setreg.c:

```
#include <windows.h>
HANDLE Stdout, Stderr;
char *cmd;
DWORD cmdlen;
void error( DWORD rc, char *msg )
  {
  DWORD bytes;
  WriteFile(Stderr, cmd, cmdlen, &bytes, NULL);
  WriteFile(Stderr, msg, strlen(msg), &bytes,
    NULL);
  ExitProcess(rc);
  }
DWORD DumpRegistry( void )
  {
  /* Dump out all the environment variables
    shown in the registry. */
  HKEY h;
  DWORD i, rc, namelen, valuelen, type, bytes;
  char name[MAX_PATH + 1], value[4096 + 2];
  rc = RegOpenKeyEx(HKEY_CURRENT_USER,
    “Environment”, 0, KEY_READ, &h);
  if (rc == ERROR_SUCCESS)
    {
    for (i = 0;  namelen = sizeof(name) - 1,
        valuelen = sizeof(value) - 2,
        (rc = RegEnumValue(h, i, name,
        &namelen, NULL, &type, value,
        &valuelen)) == ERROR_SUCCESS;  i++)
      {
      register char *t;
      for (t = name + namelen;  t < name + 12;
          *t++ = ‘ ’)
        ;
      *t++ = ‘ ’;
      WriteFile(Stdout, name, t - name,
        &bytes, NULL);
      value[valuelen++] = ‘\r’;
      value[valuelen++] = ‘\n’;
      WriteFile(Stdout, value, valuelen,
        &bytes, NULL);
      }
    if (rc == ERROR_NO_MORE_ITEMS)
      rc = 0;
    }
  return rc;
  }
DWORD DumpVariable( char *name, DWORD namelen )
  {
  HKEY h;
  char value[4096 + 2];
  DWORD i, rc, valuelen = sizeof(value) - 2,
    type, bytes;
  static char spaces[12] = “            ”;
  rc = RegOpenKeyEx(HKEY_CURRENT_USER,
    “Environment”, 0, KEY_READ, &h);
  if (rc == ERROR_SUCCESS)
    {
    rc = RegQueryValueEx(h, name, NULL, &type,
      value, &valuelen);
    if (rc == ERROR_SUCCESS)
      {
      WriteFile(Stdout, name, namelen, &bytes,
        NULL);
      i = namelen < 12 ? 13 - namelen : 1;
      WriteFile(Stdout, spaces, i, &bytes,
        NULL);
      value[valuelen++] = ‘\r’;
      value[valuelen++] = ‘\n’;
      WriteFile(Stdout, value, valuelen,
        &bytes, NULL);
      }
    }
  return rc;
  }
DWORD SetVariable( char *name, char *value,
    DWORD valuelen, DWORD type )
  {
  HKEY h;
  DWORD rc;
  rc = RegOpenKeyEx(HKEY_CURRENT_USER,
    “Environment”, 0, KEY_WRITE, &h);
  if (rc == ERROR_SUCCESS)
    rc = RegSetValueEx(h, name, 0, type, value,
      valuelen);
  return rc;
  }
void main( void )
  {
  register char *c;
  register DWORD rc;
  register BOOL got_equal;
  Stderr = GetStdHandle(STD_ERROR_HANDLE);
  if (Stderr == INVALID_HANDLE_VALUE)
    ExitProcess(GetLastError());
  cmd = c = GetCommandLine();
  while (*c && *c != ‘ ’ && *c != ‘\t’)
    c++;
  cmdlen = c - cmd;
  while (*c == ‘ ’ || *c == ‘\t’)
    c++;
  Stdout = GetStdHandle(STD_OUTPUT_HANDLE);
  if (Stdout == INVALID_HANDLE_VALUE)
    error(GetLastError(),
      “: Couldn’t open Stdout\r\n”);
  if (*c == 0)
    /* Dump out all the environment variables
      in the registry. */
    rc = DumpRegistry();
```

(Continued on page 227)

(Continued from page 226)

Figure 1. setreg.c:

```
else
  {
  register char *name = c;
  DWORD namelen;
  while (*c && *c != ' ' && *c != '\t'
      && *c != '=')
    c++;
  namelen = c - name;
  while (*c == ' ' || *c == '\t')
    c++;
  if (got_equal = *c == '=')
    {
    /* Skip over any = followed by any
       spaces. */
    c++;
    while (*c == ' ' || *c == '\t')
      c++;
    }
  if (*c == 0 && !got_equal)
    /* Dump out just the one variable
       specified. */
    rc = DumpVariable(name, namelen);
  else
    {
    /* Set the variable to the specified
       value */
    register char *t, *value = c;
    register DWORD type = REG_SZ;
    register BOOL inside_quotes = FALSE;
    /* First scan the value, fixing up any
       backslashes used as escapes, dropping
       extra whitespace and watching for
         %'s. */
    for (t = c; *c; *t++ = *c++)
      switch (*c)
        {
        case '"':
          inside_quotes = !inside_quotes;
          c++;
          break;
        case '\\':
          if (c[1] == '\\' &&
              c[2] == '"')
            c++;
          break;
        case ' ':
        case '\t':
          if (!inside_quotes)
            while (c[1] == ' ' ||
                c[1] == '\t')
              c++;
          break;
        case '%':
          type = REG_EXPAND_SZ;
        }
    *t = 0;
    name[namelen] = 0;
    rc = SetVariable(name, value, t - value,
      type);
    }
  }
ExitProcess(rc);
}
```

• HKEY_LOCAL_MACHINE, which contains information about the machine itself and the hardware and software that's been installed

• HKEY_USERS, which describes the default environment for a new user and the customized environments for existing users

• HKEY_CLASSES_ROOT, a synonym for HKEY_LOCAL_MACHINE\SOFTWARE\Classes, which lists all the known file type extensions

• HKEY_CURRENT_USER, a synonym for HKEY_USERS\<sid>, where <sid> is a long gobbledygook string giving the unique security identifier (SID), rather like a Social Security number, for a particular user. For example, on my Olivetti/MIPS machine, my SID is S-1-5-21-242430772-281344-854032384-500. There are registry functions (LookupAccountName and LookupAccountSid) for converting back and forth between user names and SIDs, but HKEY_CURRENT_USER lets you gain access to your own customization information without having to do that translation between SIDs and names.

Admittedly, we are still looking at work-in-progress in the October beta, so there are some rough spots (hopefully, by the time you read this, the next beta should be available and will most likely be more complete than this one). One such rough spot is that, while the overall architecture of the registry is fairly clear, some of the details are not. For example, what are the individual names of all the nodes in the registry and just what information should go where? Right now, the best way to answer that is just to open up the registry with RegEdit and try to make your own guesses.

Also, the user interface to the registry is meant to be via the applets in Control Panel. Clearly, not all the applets are finished yet; for now, some changes can be made only with RegEdit. But make a change incorrectly, and you've got a system that won't boot and can't easily be fixed; following one of my own failed experiments a few months back, I had to resign myself to a complete reinstallation.

Another problem with the Control Panel applets is that they are only graphical. There are no provisions for manipulating the registry from a batch file, for example—which brings us to a discussion about the programming interface to the registry and a couple of tiny utilities

Figure 2. unsetreg.c:

```
#include <windows.h>
#include <string.h>
HANDLE Stdout, Stderr;
char *cmd;
DWORD cmdlen;
void error( DWORD rc, char *msg )
  {
  DWORD bytes;
  WriteFile(Stderr, cmd, cmdlen, &bytes, NULL);
  WriteFile(Stderr, msg, strlen(msg), &bytes,
    NULL);
  ExitProcess(rc);
  }
void main( void )
  {
  HKEY h;
  register char *c;
  register DWORD rc;
  Stderr = GetStdHandle(STD_ERROR_HANDLE);
  if (Stderr == INVALID_HANDLE_VALUE)
    ExitProcess(GetLastError());
  cmd = c = GetCommandLine();
  while (*c && *c != ' ' && *c != '\t')
    c++;
  cmdlen = c - cmd;
  while (*c == ' ' || *c == '\t')
    c++;
  if (*c == 0)
    error(ERROR_INVALID_PARAMETER,
      ": Can't unset a null list.\r\n");
  Stdout = GetStdHandle(STD_OUTPUT_HANDLE);
  if (Stdout == INVALID_HANDLE_VALUE)
    error(GetLastError(),
      ": Couldn't open Stdout\r\n");
  rc = RegOpenKeyEx(HKEY_CURRENT_USER,
    "Environment", 0, KEY_WRITE, &h);
  if (rc == ERROR_SUCCESS)
    {
    register char *name, savech;
    do
      {
      name = c;
      while (*c && *c != ' ' && *c != '\t')
        c++;
      savech = *c;
      *c++ = 0;
      RegDeleteValue(h, name);
      }
    while (savech);
    }
  ExitProcess(rc);
  }
```

for setting the environment variables you'll see every time you log in from the command line or a batch file.

Figure 1 contains setreg.c, a utility used for setting an environment variable, dumping a particular variable's value or dumping all the variables. For example, to define the INCLUDE environment variable, type the following (the equal sign is optional unless you're setting a variable to a null string):

```
setreg INCLUDE = c:\mstools\h
```

To list all the variables currently defined in the registry, type setreg. To print the value of just a single variable, type setreg INCLUDE.

The code in the main procedure is fairly standard stuff—just parsing the command line to figure out what you've requested. I've chosen to get the command-line text from GetCommandLine, parsing out the punctuation that would be added if there were spaces or quotes embedded in the argument words; an alternative would have been to use argv and to concatenate the words if we were setting a value. Also, if we're setting a variable, we'll need to watch for %-style references to other variables in the value that we're giving it so we can tell the registry those should be expanded later.

Whether we're reading or writing to the registry, the first step is always the same: We have to open a handle to the appropriate key with the desired access rights using RegOpenKeyEx. Here, we're always opening the environment key inside HKEY_CURRENT_USER.

To dump the entire environment, we iterate calls to RegEnumValue. Each call returns a new name/type/value triple. We're ignoring the type here, assuming that all the values will be printable strings: a good assumption for environment variables but not one you can make in general. After all the values have been read, RegEnumValue returns ERROR_NO_MORE_ITEMS, indicating that we've successfully read them all. To dump a specific variable by name, we use RegQueryValueEx. Finally, to set a variable, we use RegSetVariable.

Above, Figure 2 contains the companion utility, called unsetreg.c. This utility deletes variables. Once again, the first step is to open a handle to the environment key; unsetreg then iterates over the list of names it's given, calling RegDeleteValue for each.

When using setreg or unsetreg or any other registry function, do remember that it's like making changes to CONFIG.SYS under DOS: The changes don't take effect immediately. Changes related to an individual user will not take effect until you log out and then back in; changes related to the system as a whole—for example, your device driver configuration—won't take effect until you reboot. ■

Douglas Hamilton is president of Hamilton Laboratories (Wayland, Mass.) and author of the Hamilton C Shell, an advanced interactive command processor and tools package for OS/2 and Windows NT. Reach Douglas on WIX as hamilton or care of Editor at the address on page 16.