Termination handlers (Chapter 4) are a useful way to ensure that the unlock is performed.

Lock Logic Consequences

Although the file lock logic in Tables 3–1 and 3–2 is natural, it has consequences that may be unexpected and cause unintended program defects. Here are some examples.

- Suppose that process A and process B periodically obtain shared locks on a file, and process C blocks when attempting to gain an exclusive lock on the same file after process A gets its shared lock. Process B may now gain its shared lock even though C is still blocked, and C will remain blocked even after A releases the lock. C will remain blocked until all processes release their shared locks even if they obtained them after C blocked. In this scenario, it is possible that C will be blocked forever even though all the other processes manage their shared locks properly.
- Assume that process A has a shared lock on the file and that process B attempts to read the file without obtaining a shared lock first. The read will still succeed even though the reading process does not own any lock on the file because the read operation does not conflict with the existing shared lock.
- These statements apply both to entire files and to file regions.
- File locking can produce deadlocks in the same way as with mutual exclusion locks (see Chapter 8 for more on deadlocks and their prevention).
- A read or write may be able to complete a portion of its request before encountering a conflicting lock. The read or write will return `FALSE`, and the byte transfer count will be less than the number requested.

Using File Locks

File locking examples are deferred until Chapter 6, which covers process management. Programs 6–4, 6–5, and 6–6 use locks to ensure that only one process at a time can modify a file.

UNIX has *advisory* file locking; an attempt to obtain a lock may fail (the logic is the same as in Table 3–1), but the process can still perform the I/O. Therefore, UNIX can achieve locking between cooperating processes, but any other process can violate the protocol.

To obtain an advisory lock, use options to the `fcntl` function. The commands (the second parameter) are `F_SETLK`, `F_SETLKW` (to wait), and `F_GETLK`. An additional block data structure contains a lock type that is one of `F_RDLCK`, `F_WRLCK`, or `F_UNLCK` and the range.

Mandatory locking is also available in some UNIX systems using a file's `set-group-ID` and `group-execute`, both using `chmod`.

UNIX file locking behavior differs in many ways. For example, locks are inherited through an `exec` call.

The C library does not support locking, although Visual C++ does supply non-standard locking extensions.

The Registry

The registry is a centralized, hierarchical database for application and system configuration information. Access to the registry is through *registry keys*, which are analogous to file system directories. A key can contain other keys or key/value pairs, where the key/value pairs are analogous to directory names and file names. Each value under a key has a name, and for each key/value pair, corresponding data can be accessed and modified.

The user or administrator can view and edit the registry contents through the registry editor, using the `REGEDIT` command. Alternatively, programs can manage the registry through the registry API functions described in this section.

Note: Registry programming is discussed here due to its similarity to file processing and its importance in some, but not all, applications. The example will be a straightforward modification of the `lsW` example. This section could, however, be a separate short chapter. *Therefore, if you are not concerned with registry programming, skip this section.*

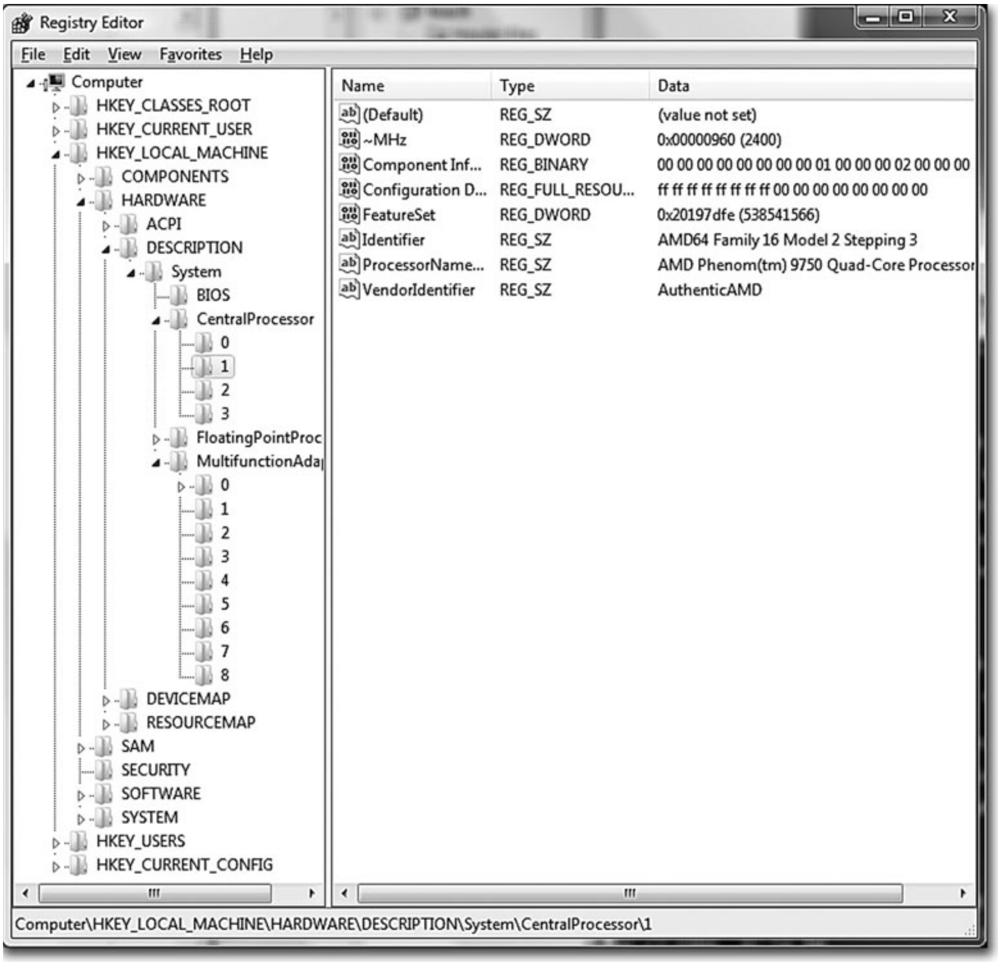
The registry contains information such as the following and is stored hierarchically in key/value pairs.

- Windows version number, build number, and registered user. However, programs usually access this information through the Windows API, as we do in Chapter 6 (the `version` program, available in the *Examples*).
- Similar information for every properly installed application.
- Information about the computer's processor type, number of processors, memory, and so on.
- User-specific information, such as the home directory and application preferences.
- Security information such as user account names.
- Installed services (Chapter 13).

- Mappings from file name extensions to executable programs. These mappings are used by the user interface shell when the user clicks on a file icon. For example, the `.doc` and `.docx` extensions might be mapped to Microsoft Word.

UNIX systems store similar information in the `/etc` directory and files in the user's home directory. The registry centralizes all this information in a uniform way. In addition, the registry can be secured using the security features described in Chapter 15.

The registry management API is described here, but the detailed contents and meaning of the various registry entries are beyond the book's scope. Nonetheless, Figure 3–1 shows a typical view from the registry editor and gives an idea of the registry structure and contents.



```
Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
  HKEY_LOCAL_MACHINE
    COMPONENTS
    HARDWARE
      ACPI
      DESCRIPTION
        System
          BIOS
          CentralProcessor
            0
            1
            2
            3
          FloatingPointProc
          MultifunctionAda
            0
            1
            2
            3
            4
            5
            6
            7
            8
      DEVICEMAP
      RESOURCEMAP
    SAM
    SECURITY
    SOFTWARE
    SYSTEM
  HKEY_USERS
  HKEY_CURRENT_CONFIG
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
~MHz	REG_DWORD	0x00000960 (2400)
Component Inf...	REG_BINARY	00 00 00 00 00 00 00 00 01 00 00 00 02 00 00 00
Configuration D...	REG_FULL_RESOU...	ff ff ff ff ff ff ff ff 00 00 00 00 00 00 00 00
FeatureSet	REG_DWORD	0x20197dfe (538541566)
Identifier	REG_SZ	AMD64 Family 16 Model 2 Stepping 3
ProcessorName...	REG_SZ	AMD Phenom(tm) 9750 Quad-Core Processor
VendorIdentifier	REG_SZ	AuthenticAMD

Computer\HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor\1

Figure 3–1 The Registry Editor

The specific information regarding the host machine's processor is on the right side. The bottom of the left side shows that numerous keys contain information about the software applications on the host computer. Notice that every key must have a default value, which is listed before any of the other key/value pairs.

Registry implementation, including registry data storage and retrieval, is also beyond the book's scope; see the reference information at the end of the chapter.

Registry Keys

Figure 3–1 shows the analogy between file system directories and registry keys. Each key can contain other keys or a sequence of values associated with a key. Whereas a file system is accessed through pathnames, the registry is accessed through keys and value names. Several predefined keys serve as entry points into the registry.

1. `HKEY_LOCAL_MACHINE` stores physical information about the machine, along with information about installed software. Installed software information is generally created in subkeys of the form `SOFTWARE\`*`CompanyName`*`\`*`ProductName`*`\`*`Version`*.
2. `HKEY_USERS` defines user configuration information.
3. `HKEY_CURRENT_CONFIG` contains current settings, such as display resolution and fonts.
4. `HKEY_CLASSES_ROOT` contains subordinate entries to define mappings from file extensions to classes and to applications used by the shell to access objects with the specified extension. All the keys necessary for Microsoft's Component Object Model (COM) are also subordinate to this key.
5. `HKEY_CURRENT_USER` contains user-specific information, including environment variables, printers, and application preferences that apply to the current user.

Registry Management

Registry management functions can query and modify key/value pairs and data and also create new subkeys and key/value pairs. Key handles of type `HKEY` are used both to specify a key and to obtain new keys.[5] Values are typed; there are several types to select from, such as strings, double words, and expandable strings whose parameters can be replaced with environment variables.

[5] It would be more convenient and consistent if the `HANDLE` type were used for registry management. There are several other exceptions to standard Windows practice that are based on Windows history.

Key Management

Key management functions allow you to open named keys, enumerate subkeys of an open key, and create new keys.

RegOpenKeyEx

The first function, `RegOpenKeyEx`, opens a named subkey. Starting from one of the predefined reserved key handles, you can traverse the registry and obtain a handle to any subordinate key.

```
LONG RegOpenKeyEx (
   HKEY hKey,
   LPCTSTR lpSubKey,
   DWORD ulOptions,
   REGSAM samDesired,
   PHKEY phkResult)
```

The parameters for this first function are explained individually. For later functions, as the conventions become familiar, it is sometimes sufficient to survey them quickly.

`hKey` identifies a currently open key or one of the predefined reserved key handles. `phkResult` points to a variable of type `HKEY` that is to receive the handle to the newly opened key.

`lpSubKey` is the subkey name you want to open. The subkey name can be a path, such as `Microsoft\WindowsNT\CurrentVersion`. A `NULL` subkey name causes a new, duplicate key for `hKey` to be opened.

`ulOptions` is reserved and must be `0`.

`samDesired` is the access mask describing the security for the new key. Access constants include `KEY_ALL_ACCESS`, `KEY_WRITE`, `KEY_QUERY_VALUE`, and `KEY_ENUMERATE_SUBKEYS`.

The return is normally `ERROR_SUCCESS`. Any other result indicates an error. Close an open key handle with `RegCloseKey`, which takes the handle as its single parameter.

RegEnumKeyEx

`RegEnumKeyEx` enumerates subkey names of an open registry key, much as `FindFirstFile` and `FindNextFile` enumerate directory contents. This function retrieves the key name, class string (rarely used), and time of last modification.

```
LONG RegEnumKeyEx (
   HKEY hKey,
   DWORD dwIndex,
   LPTSTR lpName,
   LPDWORD lpcbName,
   LPDWORD lpReserved,
   LPTSTR lpClass,
   LPDWORD lpcbClass
   PFILETIME lpftLastWriteTime)
```

`dwIndex` should be `0` on the first call and then should be incremented on each subsequent call. The value name and its size, along with the class string and its size, are returned. Note that there are two count parameters, `lpcbName` (the subkey name) and `lpcbClass`, which are used for both input and output for buffer size. This behavior is familiar from `GetCurrentDirectory` (Chapter 2), and we'll see it again with `RegEnumValue`. `lpClass` and `lpcbClass` are, however, rarely used and should almost always be `NULL`.

The function returns `ERROR_SUCCESS` or an error number.

RegCreateKeyEx

You can also create new keys using `RegCreateKeyEx`. Keys can be given security attributes in the same way as with directories and files (Chapter 15).

```
LONG RegCreateKeyEx (
   HKEY hKey,
   LPCTSTR lpSubKey,
   DWORD Reserved,
   LPTSTR lpClass,
   DWORD dwOptions,
   REGSAM samDesired,
   LPSECURITY_ATTRIBUTES lpSecurityAttributes,
   PHKEY phkResult,
   LPDWORD lpdwDisposition)
```

The individual parameters are as follows:

- `lpSubKey` is the name of the new subkey under the open key indicated by the handle `hKey`.

- lpClass is a user-defined class type for the key. Use NULL, as recommended by MSDN.
- The dwOptions flag is usually 0 (or, equivalently, REG_OPTION_NON_VOLATILE, the default). Another, mutually exclusive value is REG_OPTION_VOLATILE. Nonvolatile registry information is stored in a file and preserved when Windows restarts. Volatile registry keys are kept in memory and will not be restored.
- samDesired is the same as for RegOpenKeyEx.
- lpSecurityAttributes can be NULL or can point to a security attribute. The rights can be selected from the same values as those used with sam-Desired.
- lpdwDisposition points to a DWORD that indicates whether the key already existed (REG_OPENED_EXISTING_KEY) or was created (REG_CREATED_NEW-_KEY).

To delete a key, use RegDeleteKey. The two parameters are an open key handle and a subkey name.

Value and Data Management

These functions allow you to get and set the data corresponding to a value name.

RegEnumValue

RegEnumValue enumerates the value names and corresponding data for a specified open key. Specify an Index, originally 0, which is incremented in subsequent calls. On return, you get the string with the value name as well as its size. You also get the data and its type and size.

```
LONG RegEnumValue (
   HKEY hKey,
   DWORD dwIndex,
   LPTSTR lpValueName,
   LPDWORD lpcbValueName,
   LPDWORD lpReserved,
   LPDWORD lpType,
   LPBYTE lpData,
   LPDWORD lpcbData)
```

The data is returned in the buffer indicated by `lpData`. The result size can be found from `lpcbData`.

The data type, pointed to by `lpType`, has numerous possibilities, including `REG_BINARY`, `REG_DWORD`, `REG_SZ` (a string), and `REG_EXPAND_SZ` (an expandable string with parameters replaced by environment variables). See MSDN for a list of all the data types.

Test the function's return result to determine whether you have enumerated all the keys. The result will be `ERROR_SUCCESS` if you have found a valid key.

`RegQueryValueEx` is similar except that you specify a value name rather than an index. If you know the value names, you can use this function. If you do not know the names, you can scan with `RegEnumValue`.

RegSetValueEx

Set the data corresponding to a named value within an open key using `RegSet-ValueEx`, supplying the key, value name, data type, and data.

```
LONG RegSetValueEx (
   HKEY hKey,
   LPCTSTR lpValueName,
   DWORD Reserved,
   DWORD dwType,
   CONST BYTE * lpData,
   CONST cbData)
```

Finally, delete named values using the function `RegDeleteValue`. There are just two parameters: an open registry key and the value name, just as in the first two `RegSetValueEx` parameters.

Example: Listing Registry Keys and Contents

Program 3–4, `lsReg`, is a modification of Program 3–2 (`lsW`, the file and directory listing program); it processes registry keys and key/value pairs rather than directories and files.

Program 3–4 `lsReg`: Listing Registry Keys and Contents

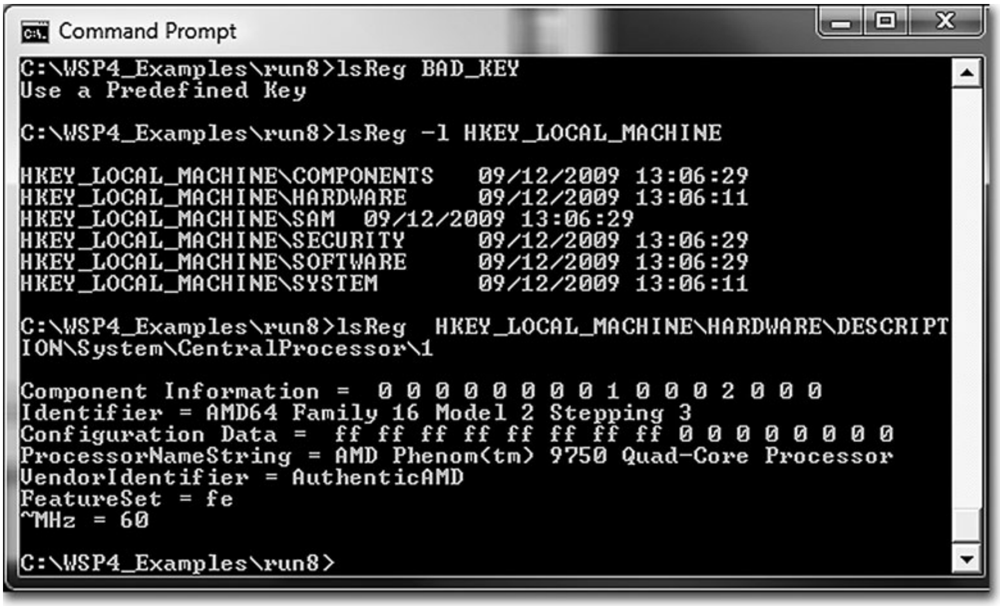
```
/* Chapter 3. lsReg: Registry list command. Adapted from Prog. 3-2. */
/* lsReg [options] SubKey */
```

```
#include "Everything.h"

BOOL TraverseRegistry(HKEY, LPTSTR, LPTSTR, LPBOOL);
BOOL DisplayPair(LPTSTR, DWORD, LPBYTE, DWORD, LPBOOL);
BOOL DisplaySubKey(LPTSTR, LPTSTR, PFILETIME, LPBOOL);

int _tmain(int argc, LPTSTR argv[])
{
   BOOL flags[2], ok = TRUE;
   TCHAR keyName[MAX_PATH+1];
   LPTSTR pScan;
   DWORD i, keyIndex;
   HKEY hKey, hNextKey;

   /* Tables of predefined key names and keys */
   LPTSTR PreDefKeyNames[] = {
      _T("HKEY_LOCAL_MACHINE"), _T("HKEY_CLASSES_ROOT"),
      _T("HKEY_CURRENT_USER"), _T("HKEY_CURRENT_CONFIG"), NULL };


HKEY PreDefKeys[] = {
      HKEY_LOCAL_MACHINE, HKEY_CLASSES_ROOT,
      HKEY_CURRENT_USER, HKEY_CURRENT_CONFIG };

   keyIndex = Options(
         argc, argv, _T("Rl"), &flags[0], &flags[1], NULL);

   /* "Parse" the search pattern into "key"and "subkey". */
   pScan = argv[keyIndex];
   for (i = 0; *pScan != _T('\\') && *pScan != _T('\0')
         && i < MAX_PATH; pScan++, i++)
               keyName[i] = *pScan;
   keyName[i] = _T('\0');
   if (*pScan == _T('\\')) pScan++;

   /* Translate predefined key name to an HKEY */
   for (i = 0; PreDefKeyNames[i] != NULL &&
         _tcscmp(PreDefKeyNames[i], keyName) != 0; i++) ;
   hKey = PreDefKeys[i];
   RegOpenKeyEx(hKey, pScan, 0, KEY_READ, &hNextKey);
   hKey = hNextKey;

   ok = TraverseRegistry(hKey, argv[keyIndex], NULL, flags);
   RegCloseKey(hKey);
   return ok ? 0 : 1;
}

BOOL TraverseRegistry(HKEY hKey, LPTSTR fullKeyName, LPTSTR subKey,
                  LPBOOL flags)
/* Traverse registry key and subkeys if the -R option is set. */
```

```
{
   HKEY hSubKey;
   BOOL recursive = flags[0];
   LONG result;
   DWORD valueType, index;
   DWORD numSubKeys, maxSubKeyLen, numValues, maxValueNameLen,
            maxValueLen;
   DWORD subKeyNameLen, valueNameLen, valueLen;
   FILETIME lastWriteTime;
   LPTSTR subKeyName, valueName;
   LPBYTE value;
   TCHAR fullSubKeyName[MAX_PATH+1];

   /* Open the key handle. */
   RegOpenKeyEx(hKey, subKey, 0, KEY_READ, &hSubKey);

   /*  Find max size info regarding the key and allocate storage */
   RegQueryInfoKey(hSubKey, NULL, NULL, NULL,
      &numSubKeys, &maxSubKeyLen, NULL, &numValues, &maxValueNameLen,
      &maxValueLen, NULL, &lastWriteTime);
   subKeyName = malloc(TSIZE * (maxSubKeyLen+1));
   valueName  = malloc(TSIZE * (maxValueNameLen+1));
   value      = malloc(maxValueLen);      /* size in bytes */

   /*  First pass for key-value pairs. */
   /*  Assumption: No one edits the registry under this subkey */
   /*  during this loop. Doing so could change add new values */
   for (index = 0; index < numValues; index++) {
      valueNameLen = maxValueNameLen + 1; /* Don't forget to set */
      valueLen     = maxValueLen + 1;     /* these values */
      result = RegEnumValue(hSubKey, index, valueName,
            &valueNameLen, NULL,
            &valueType, value, &valueLen);
      if (result == ERROR_SUCCESS && GetLastError() == 0)
         DisplayPair(valueName, valueType, value, valueLen, flags);
   }

   /* Second pass for subkeys. */
   for (index = 0; index < numSubKeys; index++) {
      subKeyNameLen = maxSubKeyLen + 1;
      result = RegEnumKeyEx(hSubKey, index, subKeyName,
            &subKeyNameLen, NULL, NULL, NULL, &lastWriteTime);
      if (GetLastError() == 0) {
         DisplaySubKey(fullKeyName, subKeyName,
            &lastWriteTime, flags);
         /*  Display subkey components if -R is specified */
         if (recursive) {
            _stprintf(fullSubKeyName, _T("%s\\%s"), fullKeyName,
               subKeyName);
            TraverseRegistry(hSubKey, fullSubKeyName,
                subKeyName, flags);
          }
       }
    }

    _tprintf(_T("\n"));
    free(subKeyName); free(valueName); free(value);
    RegCloseKey(hSubKey);
    return TRUE;
}

BOOL DisplayPair(LPTSTR valueName, DWORD valueType,
                    LPBYTE value, DWORD valueLen, LPBOOL flags)
/* Function to display key-value pairs. */
{

    LPBYTE pV = value;
    DWORD i;

    _tprintf(_T("\n%s = "), valueName);

    switch (valueType) {
    case REG_FULL_RESOURCE_DESCRIPTOR: /* 9: hardware description */
    case REG_BINARY: /*  3: Binary data in any form. */
       for (i = 0; i < valueLen; i++, pV++)
          _tprintf(_T(" %x"), *pV);
       break;
    case REG_DWORD: /* 4: A 32-bit number. */
       _tprintf(_T("%x"), (DWORD)*value);
       break;

    case REG_EXPAND_SZ: /* 2: null-terminated unexpanded string */
    case REG_MULTI_SZ: /* 7: An array of null-terminated strings */
    case REG_SZ: /* 1: A null-terminated string. */
       _tprintf(_T("%s"), (LPTSTR)value);
       break;
    /* ... Several other types */
    }

    return TRUE;
}

BOOL DisplaySubKey(LPTSTR keyName, LPTSTR subKeyName,
          PFILETIME pLastWrite, LPBOOL flags)
{
    BOOL longList = flags[1];
    SYSTEMTIME sysLastWrite;

    _tprintf(_T("\n%s"), keyName);
    if (_tcslen(subKeyName) > 0) _tprintf(_T("\\%s "), subKeyName);
```

```
   if (longList) {
      FileTimeToSystemTime(pLastWrite, &sysLastWrite);
      _tprintf(_T("%02d/%02d/%04d %02d:%02d:%02d"),
            sysLastWrite.wMonth, sysLastWrite.wDay,
            sysLastWrite.wYear, sysLastWrite.wHour,
            sysLastWrite.wMinute, sysLastWrite.wSecond);
   }
   return TRUE;
}
```

Run 3–4 shows `lsReg` operation, including using the `-l` option. The `-R` option also works, but examples require a lot of vertical space and are omitted.



```
C:\WSP4_Examples\run8>lsReg BAD_KEY
Use a Predefined Key

C:\WSP4_Examples\run8>lsReg -l HKEY_LOCAL_MACHINE

HKEY_LOCAL_MACHINE\COMPONENTS   09/12/2009 13:06:29
HKEY_LOCAL_MACHINE\HARDWARE     09/12/2009 13:06:11
HKEY_LOCAL_MACHINE\SAM  09/12/2009 13:06:29
HKEY_LOCAL_MACHINE\SECURITY     09/12/2009 13:06:29
HKEY_LOCAL_MACHINE\SOFTWARE     09/12/2009 13:06:29
HKEY_LOCAL_MACHINE\SYSTEM       09/12/2009 13:06:11

C:\WSP4_Examples\run8>lsReg  HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPT
ION\System\CentralProcessor\1

Component Information =  0 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0
Identifier = AMD64 Family 16 Model 2 Stepping 3
Configuration Data =  ff ff ff ff ff ff ff ff 0 0 0 0 0 0 0 0
ProcessorNameString = AMD Phenom(tm) 9750 Quad-Core Processor
VendorIdentifier = AuthenticAMD
FeatureSet = fe
~MHz = 60

C:\WSP4_Examples\run8>
```

Run 3–4 `lsReg`: Listing Registry Keys, Values, and Data

Summary

Chapters 2 and 3 described all the important basic functions for dealing with files, directories, and console I/O. Numerous examples show how to use these functions in building typical applications. The registry is managed in much the same way as the file system, as the final example shows.

Later chapters will deal with advanced I/O, such as asynchronous operations and memory mapping.