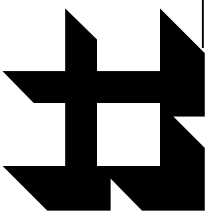

ГЛАВА 20

Работа с реестром Windows

Реестр — это один из наиболее важных аспектов Windows. И сама Windows, и различные программные объекты (приложения, серверы COM, компоненты ActiveX) хранят в реестре различную информацию. Некорректные изменения реестра могут привести к полной неработоспособности операционной системы. Тем не менее именно в реестре удобно хранить связанную с приложением информацию. Например, вы можете разместить в реестре сведения о тех параметрах вашего приложения, которые необходимо сохранять между сеансами работы, а для демонстрационных версий в нем удобно хранить счетчик запусков приложения. Кроме того, из реестра можно извлечь массу полезной информации.

В этой главе рассматриваются два способа работы с реестром; первый основан на использовании функций Windows API, второй реализуется при помощи объекта WSH (Windows Scripting Host).

Структура реестра

Информация в реестре организована в виде иерархического дерева, узлы которого принято называть ключами. Каждый ключ определяется именем и дескриптором, или целочисленным значением. Ключи могут содержать как вложенные ключи, так и конечные элементы данных. Все вложенные ключи отделяются друг от друга символом обратного слэша "\". При необходимости приложение открывает ключ реестра и использует сохраненные в нем данные. Каждый ключ реестра может содержать любое количество данных различных типов. Имена ключей не могут содержать символы пробела, звездочки "*", обратного слэша "\" и вопросительного знака. Имя вложенного ключа не должно совпадать с именами ключей, расположенных выше него в иерархии. Длина наименования ключа не может превышать 255 символов.

На вершине иерархии реестра расположены главные ключи, имеющие строго определенные имена и целочисленные значения, или дескрипторы. Главные ключи откры-

вают доступ к разделам реестра. Все вложенные ключи предоставляют доступ к подразделам внутри раздела; обращение к вложенному ключу выполняется по его имени.

В табл. 20.1 приведен список основных главных ключей реестра, значения их дескрипторов и тип хранимой "под ключом" информации.

Таблица 20.1. *Главные ключи реестра Windows*

Наименование ключа	Дескриптор	Тип хранимой информации
HKEY_CLASSES_ROOT	0x80000000	Здесь определяются классы документов и свойства, связанные с этими классами. Обычно эти свойства применяются при работе приложений, использующих технологию COM, а также приложений, использующих среду Windows (shell application). Например, когда вы в Проводнике дважды щелкаете мышью по иконке файла, система просматривает этот ключ реестра для поиска ассоциированного с файлом приложения
HKEY_CURRENT_USER	0x80000001	Здесь определяются установки, выполненные текущим пользователем, касающиеся переменных среды окружения, данных о принтерах, сетевых подключениях и т. д. Кроме того, здесь хранятся установки для приложений, используемые текущим пользователем
HKEY_LOCAL_MACHINE	0x80000002	Здесь находятся сведения о физическом состоянии компьютера, включая данные о типе процессора, оперативной памяти, установленном программном и аппаратном обеспечении
HKEY_USER	0x80000003	Здесь определяется конфигурация текущего пользователя и конфигурация по умолчанию при подключении нового пользователя
HKEY_CURRENT_CONFIG	0x80000005	Здесь описываются различия между текущей и стандартной конфигурацией

Набор главных ключей реестра может изменяться в различных версиях Windows, но перечисленные выше ключи присутствуют в каждой версии.

Для получения значения любого из ключей реестра вы должны указать полный путь к этому ключу, начиная с наименования главного ключа и перечисления всех вложенных ключей. Так, например, информация об установленном на вашем компьютере процессоре расположена в реестре по следующему адресу:

```
HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor\0
```

Запустите редактор реестра Windows (Regedit.exe) и в его окне откройте этот ключ. Окно редактора реестра должно выглядеть примерно так, как показано на рис. 20.1.

В правой части окна редактора выводится информация о значениях выбранного ключа реестра. Если ключ содержит более одного значения, то каждое значение должно

иметь собственное уникальное имя. Разрешается наличие не более одного значения, не имеющего имени. Кроме имени, каждое значение имеет тип; используемые в реестре типы данных перечислены в табл. 20.2.

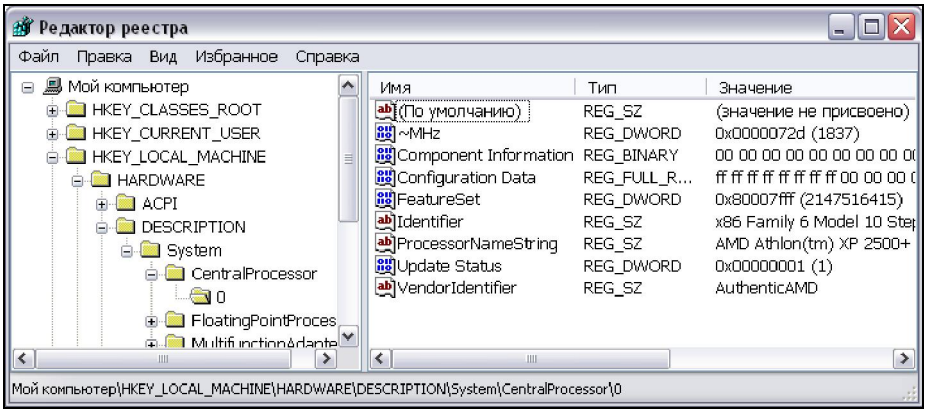

Рис. 20.1. Окно редактора реестра Windows XP

Таблица 20.2. *Типы данных реестра Windows*

Обозначение	Код	Описание
REG_NONE	0	Тип данных не устанавливается
REG_SZ	1	Нуль-терминированная строка
REG_EXPAND_SZ	2	Строка со ссылками на переменные окружения (типа %PATH%)
REG_BINARY	3	Бинарные данные в любой форме
REG_DWORD	4	Целое 32-разрядное число
REG_DWORD_BIG_ENDIAN	5	То же, что и DWORD, но порядок следования байтов изменен таким образом, что наиболее значащим является младший байт
REG_LINK	6	Символическая связь
REG_MULTY_SZ	7	Массив из нескольких строк, каждая строка заканчивается нулевым байтом; массив заканчивается двумя нулевыми байтами.

Так как мы рассматриваем реестр Windows как средство для хранения данных, необходимых вашему конкретному приложению, то нужно определиться, в каких разделах реестра целесообразно размещать эти данные.

Если информация должна быть доступна всем пользователям, ее нужно хранить в разделе, определяемом ключом HKEY_LOCAL_MACHINE. Информация для текущего пользователя должна храниться в разделе HKEY_CURRENT_USER. В каждом из этих разделов

существует вложенный ключ SOFTWARE — именно внутри этого ключа рекомендуется создавать свои вложенные ключи и размещать в них необходимую информацию. В принципе вы можете помещать данные и в другие разделы реестра, но лучше этого не делать.

Использование функций Windows API

В этом разделе мы рассмотрим основные функции Windows API, позволяющие организовать работу с реестром из вашего приложения. Эти функции реализуют следующую функциональность:

- создание нового или открытие существующего ключа реестра;
- проверка наличия ключа;
- запись, чтение и удаление значений ключа.

Открытие ключа реестра

Для того чтобы получить значения ключа, его необходимо открыть. Делается это при помощи функции RegOpenKeyEx из библиотеки Advapi.dll. Вот ее синтаксис:

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

Как видите, в прототипе функции указан параметр типа LPCTSTR, что позволяет нам предположить, что существует реализация функции с именем RegOpenKeyExA, работающая с символьными данными формата ANSI. Отметим, что все функции из библиотеки Advapi.dll имеют по две реализации, поэтому в дальнейшем, при рассмотрении других функций для работы с реестром, не будем больше акцентировать на этом ваше внимание.

Вот объявление функции RegOpenKeyEx в Visual FoxPro:

```
DECLARE Long RegOpenKeyExA IN Advapi32.dll AS RegOpenKeyEx ;
        Long hKey, String lpSubKey, Long ulOptions, ;
        Long samDesired, Long @ phkResult
```

Передаваемые функции параметры:

hKey

дескриптор ключа, расположенного в иерархии реестра непосредственно над открываемым ключом;

lpSubKey

указатель на строку, содержащую имя открываемого ключа. Если значение параметра — NULL, то открывается ключ, дескриптор которого указан в *hKey* (именно так вы можете открыть главный ключ реестра);

ulOption

целое число. Параметр зарезервирован для использования в будущем и должен быть равен нулю;

samDesired

целое число, содержит маску, определяющую доступ к ключу;

phkResult

указатель на переменную, в которую будет помещен дескриптор открываемого ключа.

Вы можете установить ограничения доступа к ключу реестра, указав в параметре *samDesired* значение битовых масок, показанных в табл. 20.3.

Таблица 20.3. *Определение прав доступа к ключу реестра*

Код маски	Значение	Описание
KEY_QUERY_VALUE	0x00000001	Права запрашивать данные вложенных ключей
KEY_SET_VALUE	0x00000002	Права устанавливать данные вложенных ключей
KEY_CREATE_SUB_KEY	0x00000004	Права создавать вложенные ключи
KEY_ENUMERATE_SUB_KEY	0x00000008	Права перебирать вложенные ключи
KEY_NOTIFY	0x00000010	Права изменять нотификацию
KEY_CREATE_LINK	0x00000020	Права создавать символическую связь
KEY_READ	0x00020019	Доступ на чтение
KEY_WRITE	0x00020006	Доступ на запись

В случае успешного выполнения функция возвращает ноль.

Чтение значений ключа

Наличие ключа в реестре — это уже информация. Но ключ может содержать различные значения, прочитать которые можно при помощи функции RegQueryValueEx. Вот ее прототип:

```
LONG RegQueryValueEx(
   HKEY hKey,
   LPCTSTR lpValueName,
   LPDWORD lpReserved,
   LPDWORD lpType,
   LPBYTE lpData,
   LPDWORD lpcbData
   );
```

Вот как выглядит объявление этой функции в Visual FoxPro:

```
DECLARE Long RegQueryValueExA IN Advapi32.dll AS RegQueryValueEx ;
        Long hkey, String lpValue, Long lpReserved, ;
        Long @ lpType, String @ lpData, Long @ lpcbData
```

Параметры функции:

hKey

дескриптор ключа;

lpValueName

указатель на строку, содержащую имя значения. Если параметр — NULL, то считывается значение и тип неименованного значения (в окне редактора реестра это значение по умолчанию — см. рис. 20.1);

lpReserved

указатель на целое число. Параметр зарезервирован; его значение должно быть равным нулю;

lpType

указатель на переменную, в которую будет помещен код типа данных (список кодов типов данных реестра см. в табл. 20.2);

lpData

указатель на буфер, в который будут записаны данные значения ключа;

lpcbData

указатель на переменную, в которую помещается фактическое значение длины буфера *lpData* в байтах.

Функция возвращает ноль при успешном выполнении. Если функция возвратит значение 234 (ERROR_MORE_DATA), то это означает, что размер выделенного буфера для размещения значения данных ключа переполнен (это возможно для типов REG_SZ и REG_MULTY_SZ).

Если тип данных значения — символьный (например, типа REG_SZ), то функция дописывает в конец сформированного значения нулевой байт, и параметр *lpcbData* содержит значение длины буфера с учетом этого дополнительного байта. Вы должны учитывать это при определении длины буфера, выделяемого для размещения возвращаемого значения.

Закрытие ключа и сохранение реестра

Если вы открыли какой-либо ключ реестра, то вы должны его обязательно закрыть. Так же не следует долго держать ключ открытым, потому что ни одно другое приложение не может обратиться к открытому вами ключу. Закрывает ключ функция RegCloseKey. Вот ее прототип:

```
LONG RegCloseKey(HKEY hKey);
```

где *hKey* — дескриптор закрываемого ключа.

Объявление функции в Visual FoxPro:

```
DECLARE Long GegCloseKey IN Advapi32.dll Long hKey
```

При обращении к реестру информация из него кэшируется в памяти процесса. Поэтому реальное обновление информации на диске может произойти не сразу после выполнения функции `RegCloseKey`, а спустя какое-то, иногда значительное, время. Принудительно сбросить изменения на диск можно при помощи функции `RegFlushKey`. В процессе выполнения эта функция использует много ресурсов и выполняется достаточно медленно, поэтому в документации по Windows ее использование не рекомендуется. В любом случае, после завершения вашего приложения все сделанные в реестре изменения будут сохранены на диске.

Пример чтения информации из реестра

Чтобы понять особенности применения рассмотренных выше функций для работы с реестром Windows, попробуем получить сведения об установленном на компьютере процессоре, в частности, узнаем имя и тип процессора, его идентификатор и тактовую частоту. Для этого нам необходимо последовательно открыть ключи реестра `HKEY_LOCAL_MACHINE`, `HARDWARE`, `DESCRIPTION`, `System`, `CentralProcessor`, и, наконец, узел 0 (см. рис. 20.1). По ходу мы должны запоминать дескрипторы открываемых узлов для того, чтобы потом правильно их закрыть.

Из последнего открытого ключа мы должны поручить данные из значений, имеющих следующие имена:

- ~MHz — частота процессора;
- ProcessorNameString — тип процессора;
- Identifier — идентификатор процессора.

Затем мы должны закрыть все открытые ключи, начиная с открытого последним.

В листинге 20.1 показан код примера, решающего поставленную задачу.

```
#DEFINE HKEY_LOCAL_MACHINE 0x80000002
#DEFINE KEY_READ           0x00020019

DECLARE Long RegOpenKeyExA IN Advapi32.dll AS RegOpenKeyEx ;
        Long, String, Long, Long, Long @
DECLARE Long RegCloseKey IN Advapi32.dll Long
DECLARE Long RegQueryValueExA IN Advapi32.dll AS RegQueryValueEx ;
        Long, String, Long, Long @, String @, Long @
*
* Создаем массив для хранения дескрипторов ключей
*
DIMENSION aHKey[4]
```

```
STORE 0 TO aHKey
*
* Открываем ключ HKEY_LOCAL_MACHINE
*
nHKey = 0
nReturn = RegOpenKeyEx(HKEY_LOCAL_MACHINE, NULL, 0, KEY_READ, @nHKey)
IF nReturn = 0
*
* Открываем ключ HARDWARE
*
    nHKey = 0
    nReturn = RegOpenKeyEx(HKEY_LOCAL_MACHINE, 'HARDWARE', 0, ;
                           KEY_READ, @nHKey)
ENDIF
IF nReturn = 0
    aHKey[1] = nHKey
*
* Открываем ключ DESCRIPTION
*
    nHKey = 0
    nReturn = RegOpenKeyEx(aHKey[1], 'DESCRIPTION', 0, KEY_READ, @nHKey)
ENDIF
IF nReturn = 0
    aHKey[2] = nHKey
*
* Открываем ключ System
*
    nHKey = 0
    nReturn = RegOpenKeyEx(aHKey[2], 'System', 0, KEY_READ, @nHKey)
ENDIF
IF nReturn = 0
    aHKey[3] = nHKey
*
* Открываем ключ CentralProcessor
*
    nHKey = 0
    nReturn = RegOpenKeyEx(aHKey[3], 'CentralProcessor', 0, ;
                           KEY_READ, @nHKey)
ENDIF
IF nReturn = 0
    aHKey[4] = nHKey
*
* Открываем ключ 0
*
    nHKey = 0
    nReturn = RegOpenKeyEx(aHKey[4], '0', 0, KEY_READ, @nHKey)
ENDIF
IF nReturn = 0
*
* Теперь выбираем данные значений ключа
*
    nLenFrq = 5
```

```
    cFrq = SPACE(nLenFrq)
    nType = 0
    nReturn = RegQueryValueEx(nHKey, '~MHz', 0, @nType, @cFrq, @nLenFrq)
    nLenCPUName = 255
    cCPUName = SPACE(nLenCPUName)
    nType = 0
    nReturn = RegQueryValueEx(nHKey, 'ProcessorNameString', 0, ;
                              @nType, @cCPUName, @nLenCPUName)
    nLenId = 255
    cId = SPACE(nLenId)
    nType = 0
    nReturn = RegQueryValueEx(nHKey, 'Identifier', 0, @nType, ;
                              @cId, @nLenId)
*
* Выводим информацию
*
    = MESSAGEBOX('Процессор: '+LEFT(cCPUName, nLenCPUName-1)+CHR(13)+ ;
                 'Идентификатор: '+LEFT(cId, nLenId-1)+CHR(13)+;
                 'Частота '+ STR(CTOBIN(LEFT(cFrq, nLenFrq), '4RS')))

    RegCloseKey(nHKey)      && Закрываем последний открытый ключ
ENDIF
*
* Закрываем все предшествующие ключи
*
FOR i = 4 TO 1 STEP -1
    IF aHKey[i] != 0
        RegCloseKey(aHKey[i])
    ENDIF
ENDFOR
IF nReturn = 0
*
* Закрываем главный узел
*
    RegCloseKey(HKEY_LOCAL_MACHINE)
ELSE
    = MESSAGEBOX('Ошибка: '+STR(nReturn))
ENDIF
```

Этот код написан, что называется, "в лоб", но он дает полное представление об использовании реестра. Обратите внимание на то, что здесь не анализируются типы возвращаемых функцией `RegQueryValueEx` данных (параметр *`nType`*), т. к. нам заранее известно, какой тип данных имеют запрашиваемые значения. Так же хотим обратить ваше внимание на то, что числовое значение параметра *`cFrq`* возвращается как строка, поэтому для него необходимо выполнить преобразование в целое число при помощи встроенной функции `CTOBIN()`.

Создание ключа

Перед тем как создать в реестре новый ключ, вы должны открыть все ключи, предшествующие ему в иерархии ключей реестра. После сознания нового ключа вы можете записать в него произвольное число значений.

Создается новый ключ функцией `RegCreateKeyEx`. Вот ее прототип:

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

Объявление в Visual FoxPro:

```
DECLARE Long RegCreateKeyExA IN Advapi32.dll AS RegCreateKeyEx ;
        Long hKey, String lpSubKey, Long Reserved, String @ lpClass, ;
        Long dwOptions, Long samDesired, String lpSecurityAttributes, ;
        Long @phkResult, Long @ lpdwDisposition
```

Параметры функции:

hKey

дескриптор ключа, в котором вы хотите создать свой ключ;

lpSuKey

указатель на строку, содержащую имя создаваемого ключа;

Reserved

целое число, зарезервированное для дальнейшего использования. Должно быть равно нулю;

lpClass

указатель на строку, в которой определен класс создаваемого ключа. Используется при создании ключей, регистрирующих объекты. Для обычных ключей — `NULL`;

dwOptions

определяет опции создаваемого ключа. Должен быть равен нулю;

samDesired

определяет маску доступа к ключу (см. табл. 20.3);

lpSecurityAttributes

указатель на структуру SECURITY_ATTRIBUTES, определяющую атрибуты безопасности создаваемого ключа. Ознакомиться с описанием структуры вы можете в MSDN. Если вы не хотите управлять безопасностью, передайте NULL как значение параметра;

phkResult

указатель на переменную, в которую будет записан дескриптор созданного ключа;

lpdwDisposition

указатель на переменную, в которую будет помещен результат выполнения функции. Если ключ создан успешно, то значение параметра равно единице. Если ключ с таким именем, как и создаваемый, уже существует, то параметр получит значение 2, а существующий ключ будет открыт.

В случае успешного завершения функция возвращает ноль.

Запись и удаление значений ключа

Возможно, вашему приложению достаточно будет просто проверять наличие в реестре нужного ключа, но, скорее всего, вы захотите поместить в этот ключ одно или несколько значений.

Для записи значений ключа применяется функция RegSetValueEx. Вот ее прототип:

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

Объявление в Visual FoxPro:

```
DECLARE Long RegSetValueExA IN Advapi32.dll AS RegSetValueEx ;
        Long hKey, String lpValueName, Long Reserved, ;
        Long dwType, String lpData, Long cbData
```

Параметры функции:

hKey

дескриптор ключа, в который нужно записать значение;

lpValueName

указатель на строку, содержащую наименование значения. Если значение параметра — NULL, то создается значение по умолчанию;

Reserved

зарезервировано для дальнейшего использования. Должно быть равно нулю;

dwType

тип данных значения (см. табл. 20.2);

lpData

указатель на буфер, в котором находится значение, помещаемое в реестр;

cpData

длина буфера *lpData* в байтах.

В листинге 20.2 показан пример создания ключа ASampleKey и добавления в него числового (имеющего тип REG_DWORD) значения и строки символов (тип REG_SZ).

```
#DEFINE HKEY_LOCAL_MACHINE 0x80000002
#DEFINE KEY_WRITE          0x00020006
#DEFINE KEY_READ           0x00020019
#DEFINE REG_SZ             0x00000001
#DEFINE REG_DWORD          0x00000004

DECLARE Long RegSetValueExA IN Advapi32.dll AS RegSetValueEx ;
        Long, String, Long, Long, String, Long
DECLARE Long RegCreateKeyExA IN Advapi32.dll AS RegCreateKeyEx ;
        Long, String, Long, String @, Long, Long, String, Long @, Long @
DECLARE Long RegOpenKeyExA IN Advapi32.dll AS RegOpenKeyEx ;
        Long, String, Long, Long, Long @
DECLARE Long RegCloseKey IN Advapi32.dll Long
DECLARE Long RegQueryValueExA IN Advapi32.dll AS RegQueryValueEx ;
        Long, String, Long, Long @, String @, Long @
*
* Открываем ключ HKEY_LOCAL_MACHINE
*
nHKey = 0
nReturn = RegOpenKeyEx(HKEY_LOCAL_MACHINE, NULL, 0, KEY_READ, @nHKey)
IF nReturn = 0
*
* Открываем ключ SOFTWARE
*
    nHKeySW = 0
    nReturn = RegOpenKeyEx(HKEY_LOCAL_MACHINE, 'SOFTWARE', 0, ;
                           KEY_WRITE, @nHKeySW)
ENDIF
IF nReturn = 0
*
* Создаем ключ ASampleKey — такое имя поместит ключ в начало ветви
* реестра
*
    nHMyKey = 0
    nResult = 0
    nReturn = RegCreateKeyEx(nHKeySW, 'ASampleKey', 0, NULL, 0, ;
                             KEY_WRITE, NULL, @nHMyKey, @nResult)
ENDIF
```

```
IF nReturn = 0
    nMyVar = 2006
    cMyText = 'Пример записи значений ключа'
    nLenMyText = LEN(cMyText)
    nReturn = RegSetValueEx(nHMyKey, 'Числовое значение', ;
                            0, REG_DWORD, BINTOC(nMyVar, '4RS'), 4)
    IF nReturn = 0
        nReturn = RegSetValueEx(nHMyKey, 'Символьная строка', ;
                                0, REG_SZ, cMyText, nLenMyText)
    ENDIF
        RegCloseKey(nHMyKey)
ENDIF
IF nHKeySW != 0
    RegCloseKey(nHKeySW)
ENDIF
IF nReturn = 0
    RegCloseKey(HKEY_LOCAL_MACHINE)
ELSE
    = MESSAGEBOX('Ошибка номер ' + STR(nReturn))
ENDIF
```

На рис. 20.2 показан вид окна программы Regedit.exe, в котором показан добавленный в результате выполнения кода ключ и его значения.

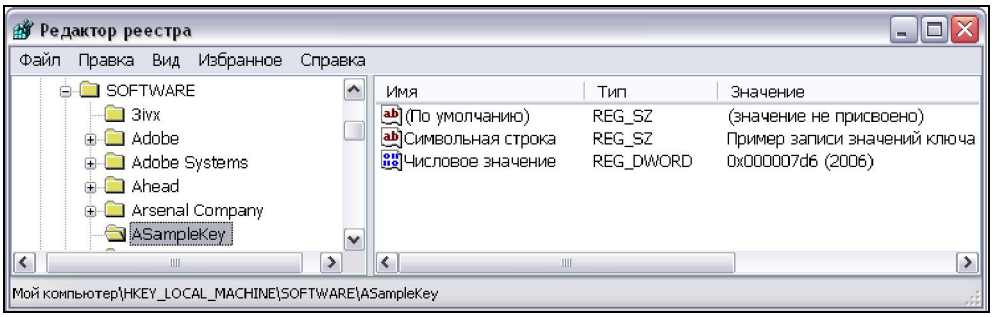

Рис. 20.2. Окно реестра после выполнения кода примера

Функция `RegSetValueEx` может использоваться также и для изменения данных существующего значения ключа.

Удаление ключа и всех связанных с ним значений из реестра выполняет функция `RegDeleteKeyEx`. Вот ее прототип:

```
LONG RegDeleteKeyEx(
    HKEY hKey,
    LPCTSTR lpSubKey,
    REGSAM samDesired,
    DWORD Reserved
);
```

Объявление в Visual FoxPro:

```
DECLARE Long RegDeleteKeyExA IN Advapi32.dll AS RegDeleteKeyEx ;
        Ling hKey, String lpSubKey, Long samDesired, Long Reserved
```

Параметры функции:

hKey

дескриптор ключа, предшествующего в иерархии удаляемому ключу;

lpSubKey

указатель на строку, содержащую наименование удаляемого ключа;

samDesired

для 32-разрядных Windows равен 0x0200, для 64-разрядных — 0x0100;

Reserved

целое число, зарезервированное для дальнейшего использования. Параметр должен быть равен нулю.

В случае успешного завершения функция возвращает ноль.

Работа с этой функций так же требует открытия всех ключей, иерархически предшествующих удаляемому ключу. В листинге 20.3 показан код, удаляющий из реестра созданный при выполнении кода листинга 20.2 ключа `ASampleKey`.

```
#DEFINE HKEY_LOCAL_MACHINE 0x80000002
#DEFINE KEY_READ           0x00020019

DECLARE Long RegOpenKeyExA IN Advapi32.dll AS RegOpenKeyEx ;
        Long, String, Long, Long, Long @
DECLARE Long RegDeleteKeyExA IN Advapi32.dll AS RegDeleteKeyEx ;
        Long, String, Long, Long
DECLARE Long RegCloseKey IN Advapi32.dll Long
*
* Открываем ключ HKEY_LOCAL_MACHINE
*
nHKey = 0
nReturn = RegOpenKeyEx(HKEY_LOCAL_MACHINE, NULL, 0, KEY_READ, @nHKey)
IF nReturn = 0
*
* Открываем ключ SOFTWARE
*
    nHKeySW = 0
    nReturn = RegOpenKeyEx(HKEY_LOCAL_MACHINE, 'SOFTWARE', 0, ;
                           KEY_READ, @nHKeySW)
ENDIF
IF nReturn = 0
*
* Удаляем из реестра ключ ASampleKey и все его значения
*
    nReturn = GegDeleteKeyEx(nHKeySW, 'ASampleKey', 0x0200, 0)
```

```
ENDIF
IF nHKeySW != 0
*
* Закрываем ключ SOFTWARE
*
    RegCloseKey(nHKeySW)
ENDIF
IF nReturn = 0
    RegCloseKey(HKEY_LOCAL_MACHINE)
ELSE
    = MESSAGEBOX('Ошибка ' + STR(nReturn))
ENDIF
```

Другие API-функции для работы с реестром Windows

Мы рассмотрели только шесть из нескольких десятков функций Windows API, предназначенных для работы с реестром. Надеемся, что предоставляемая ими функциональность окажется вполне достаточной для удовлетворения требований вашего приложения. Но если вы хотите узнать больше — читайте MSDN. В MSDN для Visual Studio 2005 всю необходимую информацию можно найти, открыв следующие узлы:

```
Win32 and COM Development
  System Services
    Windows System Information
      SDK Documentation
        Windows System Information
          Registry
            Registry Reference
              Registry Function
```

Использование Windows Scripting Host

Начиная с Windows 98, Microsoft предоставляет новую функциональность — Windows Scripting Host (WSH), или поддержку сценариев. Сценарии похожи на широко использовавшиеся в MS-DOS командные (BAT) файлы, но, в отличие от таких файлов, сценарии пишутся на специально разработанных языках VBScript и JScript. Сценарий может принимать решения на основе использования полноценных операторов if/else; он может выполнять один набор команд, если данное условие истинно, или другой набор, если условие ложно. Из одного сценария можно вызывать другие сценарии; кроме того, JScript и VBScript поддерживают выполнение математических операций, включая общие тригонометрические функции.

Возможности сценариев достаточно ограничены. Языки написания сценариев не имеют прямых инструкций, позволяющих читать или записывать файлы на диске, запускать приложения, изменять записи в реестре Windows и т. д., но в них включена поддержка технологии COM, что позволило разработчикам расширить возможности

сценариев за счет включения в Windows Scripting Host нескольких специализированных COM-объектов.

Для работы с реестром Windows используется объект `WshShell`, являющийся встроенным компонентом Windows Scripting Host. Помимо методов, позволяющих выполнять операции с записями реестра, он реализует также некоторые иные функции, которые мы рассмотрим в заключительной части этой главы.

Создание разделов и запись значений ключей реестра

Для записи информации в реестр используется метод `RegWrite` объекта `WshShell`. Методу передаются три обязательных параметра.

В первом параметре может быть передано как имя ключа для создания нового раздела, так и имя значения ключа, поэтому метод должен определить, что записать в реестр: ключ или имя значения. Для решения этой проблемы используются следующие соглашения:

- если строка параметра кончается символом "\", то это — ключ;
- отсутствие в конце строки символа "\" означает, что это имя значения ключа (например, "HKCU\MyKey\Name").

В параметре должна быть указана последовательность всех предшествующих ключей, начиная с главного ключа раздела. Разрешается использовать как полные, так и сокращенные наименования главных ключей. Регистр символов при создании ключей и имен значений сохраняется.

Следующий код создаст в разделе `HKEY_CURRENT_USER` новый ключ с именем `MYKEY`:

```
oWshShell = CREATEOBJECT("Wscript.Shell")
oWshShell.RegWrite("HKCU\MYKEY\","","REG_SZ")
```

Обратите внимание на наличие завершающего символа "\" после имени ключа.

Второй параметр метода определяет собственно значение ключа, а третий — тип данных (см. табл. 17.2). В следующем фрагменте кода показано, как добавить два новых значения к созданному выше ключу `MYKEY`:

```
oWshShell.RegWrite("HKCU\MYKEY\PARAM1","Это тест","REG_SZ")
oWshShell.RegWrite("HKCU\MYKEY\PARAM2",1000,"REG_DWORD")
```

При выполнении этого кода ключу `MYKEY` будет присвоено два значения; первое, с именем `PARAM1`, является строкой, а второе, с именем `PARAM2` — целым числом. На рис. 20.3 показано, как будет выглядеть окно редактора реестра после добавления нового раздела и присваивания значений ключу раздела.

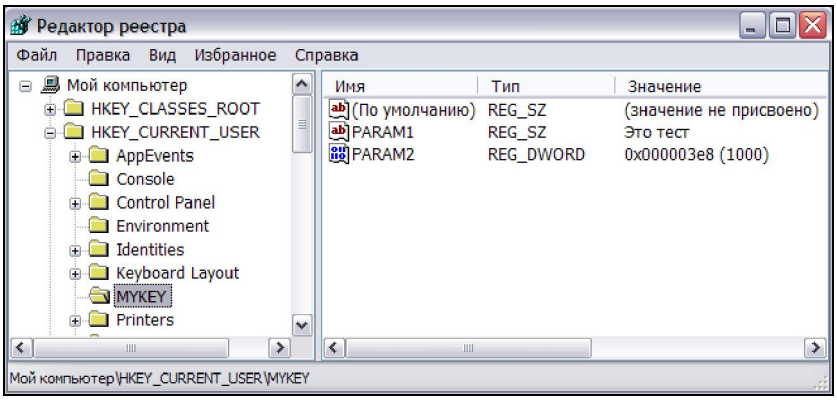

Рис. 20.3. Окно редактора реестра для ключа `MYKEY`

Чтение информации из реестра

Для чтения информации из реестра используется метод `RegRead` объекта `WshShell`. Метод возвращает указанное значение ключа. Если имя значения не указано (строка параметра заканчивается обратным слэшем), то возвращается значение ключа по умолчанию. В случае отсутствия в реестре соответствующего ключа или значения с указанным именем возникает исключение.

В следующем примере показано, как получить значения созданного нами ранее ключа `MYKEY`:

```
? oWshShell.RegRead("HKCU\MYKEY\PARAM1") && "Это тест"
? oWshShell.RegRead("HKCU\MYKEY\PARAM2") && 1000
```

При чтении информации из реестра регистр символов имен ключей и значений не учитывается.

Удаление информации из реестра

Метод `RegDelete` объекта `WshShell` удаляет ключ (и все его значения) или указанное значение ключа из реестра.

Если удаляемый ключ содержит вложенные ключи, то предварительно необходимо удалить эти ключи, иначе при попытке удаления возникнет исключение. Например, если вы создали несколько новых ключей, то удалять их нужно в обратном порядке.

В следующем примере в разделе `HKEY_CURRENT_USER\Software` создаются новые, вложенные друг в друга, ключи `Key1`, `Key2` и `Key3`:

```
oWshShell.RegWrite("HKCU\Software\Key1\Key2\Key3\","","REG_SZ")
```

Вот как нужно правильно удалять эти ключи:

```
oWshShell.RegDelete("HKCU\Software\Key1\Key2\Key3\")
oWshShell.RegDelete("HKCU\Software\Key1\Key2\")
```

```
oWshShell.RegDelete("HKCU\Software\Key1\")
```

Если вы хотите удалить одно из значений ключа, то сделать это можно, указав его имя (в отличие от имени ключа, имя значения не завершается обратным слэшем):

```
oWshShell.RegDelete("HKCU\MYKEY\PARAM1")
```

Изменение значения ключа

Как это ни странно, но для изменения значения ключа также используется метод `RegWrite` объекта `WshShell`. Чтобы изменить значения ключа, вы должны вызвать этот метод и передать ему новое значение:

```
nKey = oWshShell.RegRead("HKCU\MYKEY\PARAM2")
nKey = nKey — 1
oWshShell.RegWrite("HKCU\MYKEY\PARAM2",nKey,"REG_DWORD")
```

> ***ЗАМЕЧАНИЕ***
>
> Существует одна особенность: уникальность значения ключа определяется не только его именем, но и типом хранимых данных. То есть при изменении указанного значения ключа его тип должен оставаться неизменным, иначе для ключа просто будет создано новое значение с тем же именем, но другим типом данных.

Как видите, работа с реестром при использовании объекта `WshShell` намного проще, чем при использовании функций Windows API. Впрочем, решение об использовании того или иного способа остается за вами.

Дополнительные возможности объекта *WshShell*

Хотя изложенное далее не совсем соответствует теме главы, тем не менее, начав обсуждать функциональность объекта `WshShell`, нельзя не остановиться на некоторых весьма полезных областях его применения.

Запуск приложения

Метод `Exec` объекта `WshShell` запускает приложение и делает его окно активным. В следующем примере показано, как использовать этот метод для запуска Калькулятора:

```
oWshShell = CREATEOBJECT("wscript.shell")
oWshShell.Exec("Calc")
```

Несколько большие возможности предоставляет метод `Run`. Он не только запускает приложение Windows, но и ожидает его завершения. Вот его синтаксис:

```
object.Run(strCommand, [intWindowStyle], [bWaitOnReturn])
```

Первый параметр, *strCommand*, задает путь и имя запускаемого приложения; он также может содержать и передаваемую приложению команду.

Второй (необязательный) параметр, *intWindowStyle*, задает стиль окна запускаемого приложения и может принимать значения от 0 до 10. Если параметр опущен, то окно выводится в обычных размерах и получает фокус ввода.

Третий (необязательный) параметр, *bWaitOnReturn*, определяет, нужно ли ждать завершения запущенного приложения. Если его значение — "ложь" (или параметр опущен), то метод возвратит управление вашей программе сразу после запуска приложения. Если значение параметра — "истина", то метод будет ожидать завершения процесса.

В следующем примере при помощи метода `Run` запускается приложение dcu32.exe и выполнение программы приостанавливается до его завершения.

```
oWshShell = CREATEOBJECT("wscript.shell")
oWshShell.Run("c:\dcu\dcu32.exe", 1, .t.)
```

Имитация нажатия клавиш клавиатуры

Метод `AppActivate` объекта `WshShell` активизирует окно приложения по его имени. Активное окно получает фокус ввода; при помощи метода `SendKeys` активному окну можно посылать команды, имитирующие нажатие клавиш клавиатуры.

В листинге 20.4 показан код, запускающий приложения Калькулятор и Блокнот. В этом коде имитируются нажатия на кнопки калькулятора; результат вычислений запоминается в буфере обмена и выводится в окне Блокнота. Через три секунды после вывода результата в окно Блокнота завершается приложение Калькулятор, а затем — приложение Блокнот. Для корректного выполнения кода примера установите русскоязычную раскладку клавиатуры.

```
LOCAL loCalcTitle, loNotePadTitle, loWshShell
* Создаем объект WshShell для вызова его методов Run и AppActivate
loCalcTitle = "Калькулятор"        && Наименование окна для Калькулятора
loNotePadTitle = "Безымянный"      && Наименование окна Блокнота
loWshShell = CREATEOBJECT("wscript.shell")
* Запускаем последовательно приложения Calc и NotePad
loWshShell.Run("calc.exe",1)       && Запускаем Калькулятор
= INKEY(0.5)                       && Пауза — ожидаем окончания загрузки
loWshShell.Run("Notepad.exe",1)    && Запускаем Блокнот
= INKEY(0.5)                       && Пауза — ожидаем окончания загрузки
* Активируем окно калькулятора и передаем ему фокус ввода с клавиатуры
loWshShell.AppActivate(loCalcTitle)
= INKEY(0.5)                       && Делаем паузу для того, чтобы
                                   && калькулятор получил фокус
loWshShell.SendKeys("20")          && Выполняем некоторые вычисления
loWshShell.SendKeys("{+}")
```

```
loWshShell.SendKeys("30")
loWshShell.SendKeys("*5")          && 20 + 30 * 5
loWshShell.SendKeys("~")           && ENTER
loWshShell.SendKeys("^(C)")        && CTRL+C — результат в буфер обмена
* Активируем окно NotePad и передаем ему фокус ввода с клавиатуры
loWshShell.AppActivate(loNotePadTitle)
= INKEY(0.5)                       && Делаем паузу для того, чтобы
                                   && Блокнот получил фокус
loWshShell.SendKeys("{= 60}")      && Выводим строку из 60 символов "="
loWshShell.SendKeys("~")           && ENTER
loWshShell.SendKeys("Результат ")
loWshShell.SendKeys("^(V)")        && CTRL+V — вставка из буфера обмена
= INKEY(3)                         && Наслаждаемся результатом...
* Завершаем работу Калькулятора
loWshShell.AppActivate(loCalcTitle)
= INKEY(0.5)
loWshShell.SendKeys("%{F4}")       && ALT+F4 — для Калькулятора
* Завершаем работу NotePad
loWshShell.AppActivate(loNotePadTitle)
= INKEY(0.5)
loWshShell.SendKeys("%{F4}")       && ALT+F4 — для Блокнота
= INKEY(1)                         && Видим запрос на сохранение
loWshShell.SendKeys("н")           && Программно нажимаем "н" (Нет)
```

Метод `AppActivate` возвращает "истину", если заданное окно существует и ему передан фокус ввода.

Вы можете передавать в метод `SendKeys` строку, содержащую один или несколько символов. Существуют соглашения относительно передачи специальных символов; так, плюс, взятый в фигурные скобки, интерпретируется как символ, иначе — как признак нажатия клавиши <Shift>. Символ перевода каретки может быть указан как тильда "~", так и в виде текстовой строки "{ENTER}". Более подробную информацию вы можете найти в MSDN.

Просмотр значений переменных окружения Windows

Оболочка Windows предоставляет ряд системных переменных, в которых содержатся различные значения, анализируя которые вы можете управлять выполнением своего приложения. Объект `WshShell` размещает эту информацию в коллекции `Environment`. Используя цикл `FOR EACH`, вы можете просмотреть содержимое этой коллекции, как показано в листинге 20.5.

```
CLEAR
LOCAL loWshShell, loSysEnv, loItem
loWshShell = CREATEOBJECT("WScript.Shell")
loSysEnv = loWshShell.Environment("PROCESS")
FOR EACH loItem IN loSysEnv
```

```
   ? loItem
ENDFOR
```

Результат выполнения этого кода на нашем компьютере показан в листинге 20.6.

```
ALLUSERSPROFILE=C:\Documents and Settings\All Users
APPDATA=C:\Documents and Settings\Author\Application Data
CLIENTNAME=Console
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=COMPUTER
ComSpec=C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_CHECK=NO
HOMEDRIVE=C:
HOMEPATH=\Documents and Settings\Author
LOGONSERVER=\\COMPUTER
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
Path=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 6 Model 10 Stepping 0, AuthenticAMD
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=0a00
ProgramFiles=C:\Program Files
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\DOCUME~1\8C45~1\LOCALS~1\Temp
TMP=C:\DOCUME~1\8C45~1\LOCALS~1\Temp
USERDOMAIN=COMPUTER
USERNAME=Папа
USERPROFILE=C:\Documents and Settings\ Author
VS80COMNTOOLS=C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\
windir=C:\WINDOWS
```

Вместо параметра `PROCESS` для `Environment` вы можете указать `SYSTEM` (просмотр значений системных переменных) или `USER` (просмотр переменных окружения, связанных с текущим пользователем).

Существует еще один способ получения информации из переменных окружения Windows, заключающийся в использовании метода `ExpandEnvironmentStrings` объекта `WshShell`. Вы должны передать методу имя переменной окружения, ограниченное символами "`%`". В результате выполнения метод вернет значение этой переменной.

Пример использования метода `ExpandEnvironmentStrings` показан в листинге 20.7.

```
CLEAR
```

```
LOCAL loWshShell
loWshShell = CREATEOBJECT("WScript.Shell")
? loWshShell.ExpandEnvironmentStrings("%WinDir%")
? loWshShell.ExpandEnvironmentStrings("%SystemRoot%")
? loWshShell.ExpandEnvironmentStrings("%UserProfile%")
```

Заключение

Как известно, добиться одного и того же результата в Windows можно совершенно различными способами. В этой главе вы узнали, как можно работать с реестром Windows, используя Windows API-функции, и как для этой же цели использовать готовые COM-компоненты. Хочется также отметить, что объект Windows Scripting Host предоставляет много других возможностей, не рассмотренных в этой главе, многие из которых могут быть использованы в ваших приложениях. Как всегда, более подробную информацию вы можете найти в MSDN.