12 Средства конфигурирования и управления, элементы системного администрирования

12.1 Конфигурирование и настройка

12.1.1 Общие подходы

Основной принцип – отделение программ от данных (в т.ч. конфигурационных).

Возможные подходы к хранению конфигурационной информации: централизованной и децентрализованный
На практике подходы не исключают полностью друг друга и нередко комбинируются.

Децентрализованный подход – каждое отдельное приложение (группа взаимосвязанных приложений) решают задачи хранения и использования конфигурации самостоятельно и независимо от других. Способ хранения – обычно файл, формат – текстовый (текст с разметкой: .ini, .cfg, XML и т.д.).
Гибкость, универсальность, независимость, переносимость.
Критическая важность документирования структуры и форматов данных.
Возможно наличие специализированных приложений, обеспечивающих UI и API для более удобного доступа к данным; API может также быть предоставлен в виде библиотек, рассчитанных на определенный формат конфигураций (например, поддержка формата .ini-файлов Windows).

Централизованный подход – наличие единого хранилища конфигурационных данных, подконтрольного специализированной службе, которая предоставляет доступ к данным посредством соответствующего API.
Унификация доступа, разгрузка прикладных программ от функций управления конфигурациями и интерпретации форматов.
Зависимость от качества реализации системной поддержки.

12.1.2 Реестр Windows – назначение и организация

Системный реестр (***registry***) – специальная системная иерархическая база данных, в которой приложения и операционная система могут сохранять информацию:
– сведения об аппаратной конфигурации (данные Configuration manager)
– сведения об установленных программах и службах
– список драйверов и значений их параметров
– обслуживание различных административных программ (например, Control Panel)
– произвольные данные приложений пользователя

Непосредственный доступ к реестру закрыт на уровне файловой системы. Система предоставляет соответствующий API (системные функции) для действий с реестром (с контролем прав доступа).

Реестр существовал еще в Windows 3.1, но использовался очень ограниченно. В Win 32 он стал основным средством хранения конфигураций и служебных данных. Реестры различных версий Windows могут считаться совместимыми на уровне поддержки в API.

Физический уровень представления реестра – файлы. Их состав и размещение могут различаться в зависимости от версии системы. В Windows 7 это директорий

```
%windir%\System32\config\
```

и файлы **`DEFAULT`**, **`SAM`**, **`SECURITY`**, **`SOFTWARE`**, **`SYSTEM`**.
(В Win 9x имена файлов реестра имели расширение **`.DAT`**.)

Hive (букв. «улей») – двоичный образ данных реестра в структурах в памяти; загруженный в память реестр.
Hive загружается из файлов реестра и при необходимости выгружается (экспортируется) в файлы.

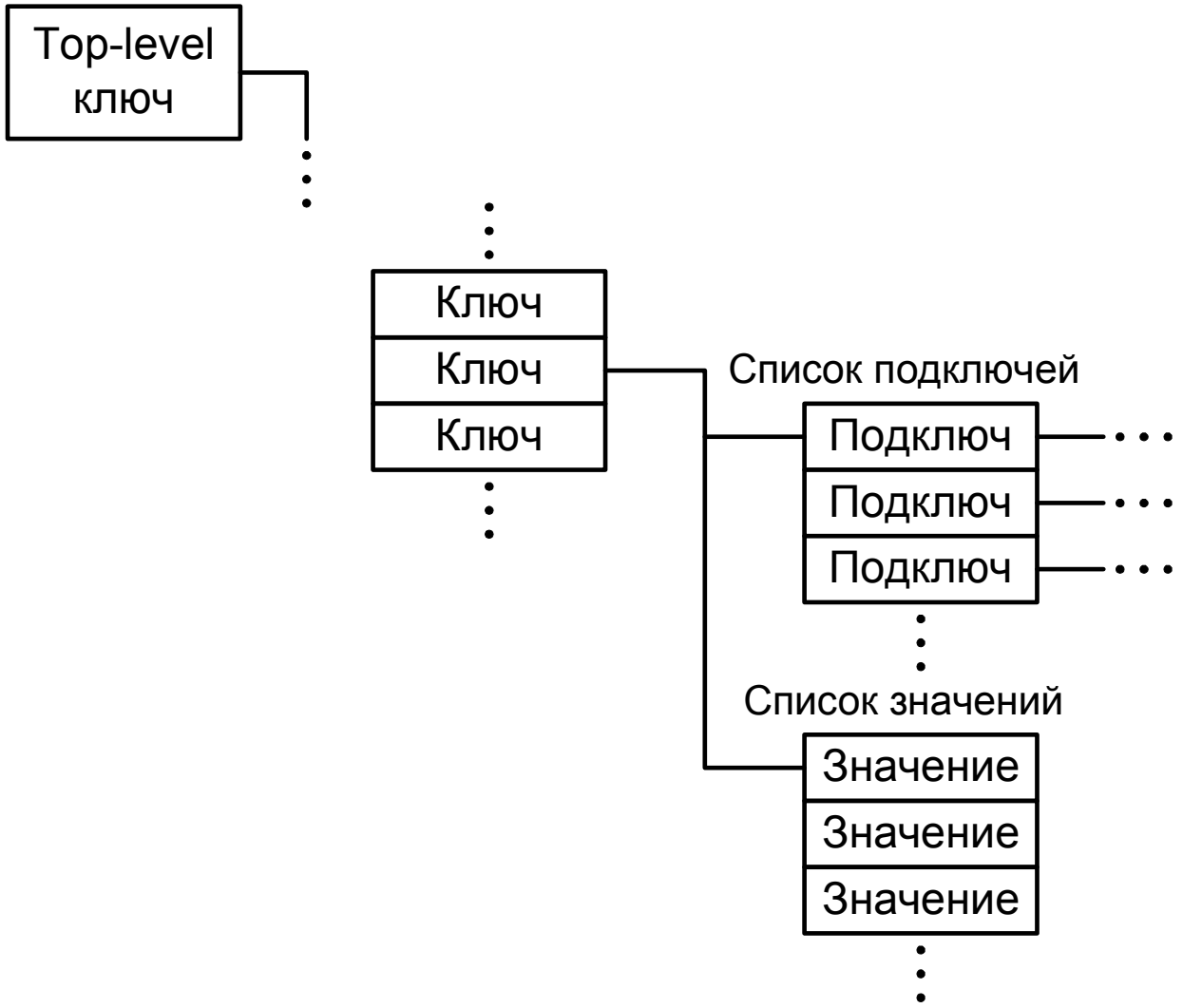
Логический уровень представления – иерархический список (дерево).

Узлы дерева двух видов:
– ***ключи*** (***keys***)
– ***значения*** (***values***).

Каждому ключу принадлежат список нижестоящих ключей (***подключей***, ***subkeys***) и список значений (возможно, пустые).

Кроме «настоящих» ключей возможны ***ссылки*** – они выглядят и используются так же, как и ключи, но фактически указывают на какой-либо уже существующий в реестре ключ.

Глубина дерева ключей ограничена – 512 уровней.



Фрагмент структуры реестра

Идентификация ключей в целом аналогична применяемой в файловой системе:
– имя (Unicode; длина до 255 символов, в т.ч. путь; регистроне-чувствительное) – в реестре, для локализации и открытия
– значение дескриптора **`HKEY`** (производный от Handle) – для доступа к уже открытому ключу

В силу особенностей реализации доступа к реестру считается, что он начинается с ***предопределенных*** (***predefined***) ключей верхнего уровня:

– **HKEY_LOCAL_MACHINE** – описание всех конфигураций, зависящих от аппаратного обеспечения компьютера

– **HKEY_USERS** – описание всех конфигураций, зависящих от пользователей компьютера

и ***ссылок*** на актуальные в текущем сеансе их подключи:

– **HKEY_CLASSES_ROOT** – ссылка на один из подключей **HKEY_LOCAL_MACHINE**, описание текущей конфигурации

– **HKEY_CURRENT_USER** – ссылка на один из подключей **HKEY_USERS**, описание конфигурации, соответствующей текущему пользователю

а также

– **HKEY_CURRENT_CONFIG** – ссылка на подключ Config в **HKEY_LOCAL_MACHINE**

В частности, приложениям рекомендуется сохранять свои данные в определенных подключах.

– Информация, зависящая от аппаратной конфигурации и общая для всех пользователей:

```
HKEY_LOCAL_MACHINE\Software\
  MyCompany\MyProduct\Version_#\...
```

– Информация, зависящая от текущего пользователя и специфичная для него

```
HKEY_CURRENT_USER\Software\
  MyCompany\MyProduct\Version_#\...
```

В 64-разрядных версиях Windows, в связи с сохранением параллельно и 32-разрядных подсистем, информация в реестре также частично дублируется – для 64- и 32-разрядной версий компонентов системы и приложений. Логическое отображение (***logical view***) реестра. Отображаемые (***reflected***), подменяемые (***redirected***) и совместно используемые (***shared***) ключи реестра.

Значения – именованные элементы данных в реестре. Идентификатор значения – имя (длина до 260 символов ASCII или 16383 символов Unicode). аналогично именам ключей). Предусмотрено наличие у ключей безымянного «значения ***по умолчанию***» («***default***»), но создавать такие значения нужно явно, как и любые другие.

Значения характеризуются ***типом***:

REG_BINARY	«двоичный блок» (массив байт)
REG_DWORD	двойное слово (машинно-зависимый формат)
REG_DWORD_LITTLE_ENDIAN	двойное слово в формате Intel (младший байт по младшему адресу)
REG_DWORD_BIG_ENDIAN	двойное слово в формате DEC (старший байт по младшему адресу)
REG_SZ	ASCIIZ-строка
REG_MULTI_SZ	массив ASCIIZ-строк, признак конца – пустая строка, т.е. два байта '\0' подряд
REG_EXPAND_SZ	ASCIIZ-строка – имя переменной окружения
REG_LINK	ссылка
REG_RESOURCE_LIST	список ресурсов (драйвера)
REG_NONE	тип не определен

Правила при использовании реестра:

– В реестр записываются только данные о конфигурации и инициализации приложения, остальные данные должны храниться в другом месте (но трактовка понятия «конфигурационных данных» не конкретизировано);

– Данные размером более 2 Кбайт должны храниться в файле, в реестр же записывается только имя и расположение этого файла (предельный размер одного значения в «стандартном формате» реестра – 1 Мбайт, но большой объем реестра сильно снижает эффективность);

– В реестре не должен храниться исполняемый код (видимо, следует распространить и на интерпретируемые скрипты);

– Данные должны быть по возможности сгруппированы в структуру (вместо отдельных ключей для каждого элемента данных), так как значения занимают меньше места, чем ключи;

– При корректном удалении (деинсталляции) приложения оно должно удалить из реестра все свои данные либо дать системе или другому (контролирующему) приложению необходимую для этого информацию.

Тем не менее, исправление ошибок и «чистка» реестра – актуальная задача. Как минимум, два основных подхода: ведение базы данных о всей последовательности модификаций реестра с возможностью восстановления предыдущих состояний, и простая сверка всех значений и ссылок на соответствие реальному содержимому диска и составу устройств в системе.

12.1.3 Пользовательский интерфейс для работы с реестром – утилита Regedit

Двоичный формат и относительно сложные правила организации реестра требуют для работы с ним специализированных программ. В системе стандартно присутствует утилита **`regedit`**, также существуют и альтернативные.

Режимы работы regedit:

Интерактивный – используя GUI (отображение дерева реестра, поиск и модификация его узлов)

Командный (***пакетный***) – выполнение «сценариев» (файлы **`.reg`**)

Роль сценариев **`regedit`** выполняют файлы данных реестра, они же участвуют в операциях импорта и экспорта. Фактически выполнение сценария соответствует его импорту в реестр.

Пример – фрагмент **.reg**-файла WinRAR:

```
REGEDIT4
[HKEY_CURRENT_USER\Software\WinRAR]
...
[HKEY_CURRENT_USER\Software\WinRAR\FileList\ArcColumn
Widths]
"name"=dword:00000203
"size"=dword:00000050
"psize"=dword:00000050
"type"=dword:00000078
"mtime"=dword:00000064
"crc"=dword:00000046
...
```

12.1.4 Программный интерфейс для работы с реестром – функции API

Функции для работы с ключами (выборочно):

Создание нового подключа уже открытого ключа:

```
LSTATUS RegCreateKey(HKEY hKey, LPCTSTR lpSubkeyName,
  PHKEY phSubkey)

LSTATUS RegCreateKeyEx( HKEY hKey,
  LPCTSTR lpSubkeyName, DWORD dwReserved,
  LPTSTR lpClass, DWORD dwOptions, REGSAM samDesired,
  LPSECURITY_ATTRIBUTES lpSecurity, PHKEY phSubkey,
  LPDWORD lpDisposition
)
```

Открытие существующего подключа уже открытого ключа:

```
LSTATUS RegOpenKey(HKEY hKey, LPCTSTR lpSubkeyName,
  PHKEY phSubkey)

LSTATUS RegOpenKeyEx( HKEY hKey,
  LPCTSTR lpSubkeyName, DWORD dwOptions,
  REGSAM samDesired, PHKEY phSubkey
)
```

Закрытие открытого ключа:

```
LSTATUS RegCloseKey( HKEY hKey)
```

Удаление подключа (по имени) открытого ключа:

```
LSTATUS RegDeleteKey( HKEY hKey,LPCTSTR lpSubkey)
```

Принудительная запись содержимого ключа из загруженного образа в файл реестра:

```
LSTATUS RegFlushKey( HKEY hKey)
```

Сохранение (экспорт) и загрузка (импорт) данных реестра, используя внешние файлы:

```
LSTATUS RegSaveKey( HKEY hKey, LPCSTR lpFileName,
  LPSECURITY_ATTRIBUTE lpSecurity );

LSTATUS RegLoadKey( HKEY hKey, LPCSTR lpSubkeyName,
  LPCSTR lpFileName );
```

Выгрузка (удаление из образа) заданного подключа:

```
LSTATUS RegUnLoadKey( HKEY hKey,
  LPCSTR lpSubkeyName );
```

Обновление содержимого ключа данными из файла – вся текущая иерархия ключа удаляется и заменяется на новую, загруженную из файла:

```
LSTATUS RegRestoreKey(HKEY hKey, LPCTSTR lpFileName,
  DWORD dwFlags );
```

```
LSTATUS RegConnectRegistry(LPTSTR lpMachine,
  HKEY hKey, PHKEY phConnected );
```

Замена файла, из которого восстанавливаются ключи, другим; эффект после перезагрузки:

```
LSTATUS RegReplaceKey( HKEY hKey, LPCSTR lpSubKey,
  LPCSTR lpNewFile, LPCSTR lpOldFile );
```

Получение списка подключей открытого ключа «перечислением» их по индексам:

```
LSTATUS RegEnumKey( HKEY hKey, DWORD dwIndex,
  LPTSTR lpSubkeyName, DWORD cbSubkeyName );

LSTATUS RegEnumKeyEx( HKEY hKey, DWORD dwIndex,
  LPTSTR lpSubkeyName, DWORD cbSubkeyName,
  LPDWORD lpReserved, LPTSTR lpClass,
  LPDWORD lpcbClass, PFILETIME lpLastWriteTime );
```

Функции для работы со значениями (выборочно):

Получение значения из открытого ключа или его подключа:

```
LSTATUS RegQueryValue( //значение подключа «по умолчанию»
  HKEY hKey, LPCTSTR lpSubkey, LPTSTR lpValue,
  PLONG lpcbValue );

LSTATUS RegQueryValueEx( //значение по имени
  HKEY hKey, LPTSTR lpValueName, LPDWORD lpReserved,
  LPDWORD lpdwType, LPTSTR lpValue, PLONG lpcbValue );
```

Список значений открытого ключа (типы и данные):

```
LSTATUS RegQueryMultipleValue(
  HKEY hKey, PVALENT pValList, DWORD dwValsNum,
  LPTSTR lpValBuf, LPDWORD lpdwTotalSize );
```

Список значений открытого ключа (имена и данные) «перечислением» их по индексам:

```
LSTATUS RegEnumValue( HKEY hKey, DWORD dwIndex,
  LPTSTR lpValName, LPDWORD lpcbValName,
  LPDWORD lpReserved, LPDWORD lpType,
  LPBYTE lpData, LPDWORD lpcbData );
```

Запись (модификация) значения:

```
LSTATUS RegSetKeyValueA(
  HKEY hKey, LPCSTR lpSubKey, LPCSTR lpValueName,
  DWORD dwType, LPCVOID lpData, DWORD cbData );

LSTATUS RegSetValueExA(
  HKEY hKey, LPCSTR lpValueName, DWORD Reserved,
  DWORD dwType, const BYTE *lpData, DWORD cbData );
```

Создание оповещения (посредством объекта синхронизации Event) об изменениях заданного значения:

```
LSTATUS RegNotifyChangeKeyValue(
  HKEY hKey, BOOL bWatchSubtree,
  DWORD  dwNotifyFilter, HANDLE hEvent,
  BOOL fAsynchronous
);
```

12.2 Протоколирование (журналирование)

12.2.1 Общие подходы

Основная идея: фиксация сведений о событиях (включая ошибки и сбои), происходящих в системе в целом, ее отдельных компонентах, в прикладных программах.

Типичные решаемые журналами задачи:

– Обнаружение и диагностика ошибок и сбоев, отладка

– Восстановление после ошибок и сбоев

– Мониторинг процессов, качественные и количественные оценки их поведения

– Сбор данных для ранней диагностики и оптимизации системы

– Контроль функционирования средств безопасности (попытки нарушения)

Важное значение журналов («логов») для отладки программ и их мониторинга, особенно в реальном («***production***») режиме работы.

Отдельно – журналируемые файловые системы и СУБД: специфические функции и требования.

Децентрализованное журналирование – приложения ведут журналы самостоятельно, по собственным правилам, обычно в форме текстовых файлов.

Централизованное журналирование – осуществляется единой службой, к которой обращаются все приложения, нуждающиеся в функциях журналирования. Возможность единообразного доступа, использования более эффективного формата хранения данных, обеспечения сохранности и целостности.

Некоторые типичные проблемы в связи с журналированием:

– Дополнительные затраты времени и других вычислительных ресурсов на запись и обслуживание журналов

– Запись в сами журналы может сопровождаться дополнительными ошибками, которые могут мешать основным функциям журналов

12.2.2 Подсистема журналирования Windows

Журналы (***Event Logs***) – в Windows NT начиная с версии 3.1 и всех последующих.
В Windows 9x поддержка отсутствовала.

Единая унифицированная технология протоколирования событий, ошибок и диагностических сообщений для всех программ в системе.

Три основных журнала:

Журнал приложений (***Application***) – используется прикладными программами

Системный журнал (***System***) – используется драйверами и службами

Журнал системы безопасности (***Security***)

Расположения файлов журнала, а также дополнительные сведения о них и об использующих их программах, содержатся в соответствующих подключах ключа реестра:

```
\HKEY_LOCAL_MACHINE\SYSTEM\
  Current Control Set\Services\EventLog
```

Событие (***Event***) – объект или событие, информация о котором заносится в журнал.

Поддерживаются 5 типов событий, идентифицируемых числовыми константами:

Сообщения об ошибках	**EVENTLOG_ERROR_TYPE**	1
Предупреждения	**EVENTLOG_WARNING_TYPE**	2
Информационные сообщения	**EVENTLOG_INFORMATION_TYPE**	4
Сообщение аудита – успешная операция	**EVENTLOG_AUDIT_SUCCESS**	8
Сообщение аудита – неуспешная операция	**EVENTLOG_AUDIT_FAILURE**	16

Данные в журналах структурированы.

Собственно журнал – идентификаторы событий и моменты их возникновения; прочие сведения хранятся в отдельных файлах (редко изменяемая информация):

Файл ***категорий*** – отражение классификации событий и особенностей их протоколирования

Файл ***сообщений*** – тексты сообщений зарегистрированных событий

Файл ***параметров*** – описание параметров зарегистрированных событий

Косвенное обращение к текстовой информации (например, строкам сообщений) упрощает в том числе и многоязыковую поддержку службы журналирования.

Удаление записей из журнала – либо явно, либо посредством лимитов (размер журнала, срок хранения записей).

Интерфейс пользовательского уровня – системная утилита:
Administrative Tools – Computer Management – Event Viewer

Программный интерфейс – несколько групп функций (API).
Основные этапы использования журнала: открытие, модификация (запись, удаление и др.), закрытие.

Открытие журнала для записи сообщений:

```
HANDLE RegisterEventSource( lpUNCServerName,
  lpSourceName );
```

Запись об очередном событии (тип и категория события, опциональная дополнительная информация; текст события уже имеется в файле сообщений):

```
ReportEvent( hEventLog, wType, wCategory, dwEventID,
  lpUserSid, wNumStrings, dwDataSize, lpStrings,
  lpRawData );
```

Закрытие журнала после записи:

```
BOOL DeregisterEventSource( hEventLog );
```

Открытие журнала для чтения и администрирования:

```
HANDLE OpenEventLog( lpUNCServerName, lpSourceName );
```

После этого на открытом журнале можно выполнять различные действия:, создание резервной копии, очистку и т.д.:

`ReadEventLog()` – чтение

`BackupEventLog()` – создание резервной копии

`ClearEventLog()` – очистка журнала

и т.д.

Закрытие журнала после администрирования:

```
BOOL CloseEventLog( hEventLog);
```

Программы могут самостоятельно регистрировать собственные события со специфическими параметрами. При этом они должны также самостоятельно зарегистрировать в системе соответствующую информацию.

Все файлы, связанные с журналом, представляют собой модули DLL, программный код которых ограничивается единственной (обязательной) пустой головной функцией, например, **`DLLMain`** (имя может зависеть от системы программирования). Эта функция возвращает значение **`TRUE`** (нормальная успешная инициализация модуля). В качестве данных к модулю присоединяется на этапе компоновки (linking) RES-файл, полученный ***компилятором сообщений*** **`MC`** из исходного текста сообщений.

Исходный текст сообщений, т.н. MC-файл, представляет собой последовательность текстовых (ASCII) строк и состоит из необязательного заголовка и последовательности блоков описания сообщений. Каждый такой блок начинается с директивы **`MessageID`**, вводящей идентификатор очередного сообщения.

В результате трансляции файла сообщений появляется также заголовочный файл C/CPP с константами – идентификаторами сообщений, который используется затем в прикладных программах.

12.3 Сценарии. Интерпретаторы сценариев («оболочки»)

Командная строка – исторически более ранний вариант пользовательского интерфейса (по сравнению с GUI), а в некоторых ОС – и единственный (классический Unix).

В Windows роль командной строки (и интерпретаторов) относительно невелика в силу особенностей идеологии системы: преобладающий графический интерфейс пользователя; централизация средств управления системой; предпочтение сложных приложений, предоставляющих комплексное решение задач. Тем не менее, в ряде случаев командная строка – более простой, эффективный и прямой способ решения задачи.

Общий вид командной строки:

<команда> <аргумент1> <аргумент2> …

Здесь ***<команда>*** – внутренняя команда интерпретатора или внешняя утилита (программа)

За ввод и выполнение команд отвечает отдельная программа – ***командный интерпретатор*** («оболочка», ***shell***). Концепция «оболочки» командной строки оформилась в ОС Unix (Bourne Shell sh и др.) и до сих пор играет важнейшую роль в большинстве Unix-систем.

Командные интерпретаторы в ОС Microsoft:

– **COMMAND.COM** (DOS, Win 16, Win 9x)

– **CMD.EXE** (Win NT и все последующие)

– PowerShell (версия 2.0 – с Windows 7, позднее 3.0 и т.д.)

Интерактивный режим – ввод команд вручную, последовательно, и поочередное их выполнение

Пакетный режим – последовательность команд в файле, который выполняется как исполняемый.
Традиционно расширение имени файла для **`COMMAND.COM`** и **`CMD.EXE`** – **`.bat`** (сокращение от «batch»).

Командный интерпретатор реализует специализированный язык программирования, предназначенный в первую очередь для управления выполнением других команд.

Базовая возможность взаимодействия порождаемых процессов – перенаправление результатов выполнения команд (потоков ввода-вывода):

> и **<** – передача данных в файл и прием данных из файла

| – конвейеризация (pipe) из команды в команду

12.3.2 PowerShell

PowerShell – расширение командного интерпретатора CMD, сочетает возможность исполнения как традиционных команд (они, впрочем, считаются псевдонимами – «alias»), так и принципиально новых – ***командлетов*** (***cmdlet***), реализует объектную технологию интерфейса, а также имеет ряд синтаксических и других отличий, например существенно лучшую поддержку арифметических и логических операций.

Раздельные 32- и 64-разрядные версии PowerShell.

Частичная совместимость версий:
– обратная совместимость версий 3, 4 и 5 с версией 2 (опция «**`-Version 2.0`**»)
– неполная совместимость версии 5 с версиями 3 и 4
– неполная совместимость при использовании разных версий CLR и MS .NET Framework

Открытый исходный текст. Наличие версий PowerShell Core для Linux и MacOS (предполагают наличие уже установленного .NET Core).

Linux: **`pwsh`**; поддерживаются дистрибутивы (как минимум) Red-Hat/CentOS/Fedora, Debian/Ubuntu, OpenSUSE, ArchLinux; для прочих установка возможна, но не гарантируется.

MacOS: **`pwsh`**; версия ОС 10.12 и выше.

ОС BSD/FreeBSD: существует проект портирования .NET Core и других компонентов, включая PowerShell.

Командлеты (***Cmdlets***)

– Унифицированное именование
– Объектная модель программного интерфейса
– Связь с Common Language Runtime (***CLR***), классами .NET

Поддерживаются интерактивный и пакетный режимы работы.

Расширение имени для пакетных файлов (скриптов) PowerShell – **`.ps1`**, **`.ps2`** и т.п.

По умолчанию, пакетный файл ассоциирован с текстовым редактором, и его выполнение заблокировано. Необходимо явно разрешить выполнение скриптов PowerShell командой в самом PowerShell, это требует полномочий администратора:

```
set-executionpolicy unrestricted
```

Некоторые полезные опции интерфейса:

– Автодополнение вводимых строк (клавиша TAB) и история команд

– Копирование текста в окно PowerShell из буфера обмена (правая кнопка «мыши»)

– Прокрутка содержимого окна PowerShell

Например:

```
get-help * > pshelp.txt
```

(Выводимый и сохраняемый в файле текст будет иметь кодировку Unicode)

Переменные:

```
$d = get-date
```

Предопределенные имена, например:
– **`$_`** – имя объекта в цикле-итераторе **`foreach`**
– **`$psversiontable`** – текущая версия PowerShell
и т.д.

Доступ к аргументам командной строки: **`$args[0]`**, **`$args[1]`** и т.д. (нумерация аргументов начинается с нуля)

Благодаря доступу к .NET и поддержке форматов данных, включая XML, обеспечивается взаимодействие с различными приложениями и системными службами.

Проблема обеспечения безопасности при использовании скриптовых языков, особенно имеющих доступ достаточно глубоко к интерфейсам системы.

12.4 Windows Management Instrumentation

Windows Management Instrumentation (***WMI***) – набор средств, «инфраструктура» (терминология Microsoft) для управления данными и функционированием Windows-системы. Технически представляет собой специфическую «фирменную» реализацию Web-Based Enterprise Management (***WBEM***) и использует Common Information Model (***CIM***). Ранее использовалась собственная технология Distributed Component Object Model (***DCOM***), в дальнейшем происходила миграция в сторону более общепринятого Simple Object Access Protocol (***SOAP***), и в версии PowerShell 3.0 присутствует альтернативный набор классов, использующих вместо DCOM основанный на SOAP протокол Windows Remote Management (***WinRM***).

Программная модель WMI – набор объектов (со своими свойствами и методами), открывающих доступ к различным компонентам системы и доступных «внешним» программам, в т.ч. и скриптам PowerShell. Таким образом, это полноценный API для управления системой.

Описания объектов хранятся в виде файлов **`.mof`** в формате ***MOF*** (Manage Object Format) в директориях:

`%windir%\System32\wbem\`
`%windir%\SysWOW64\wbem\`

(**`%windir%`** – «системный» директорий ОС, обычно **`C:\Windows\`**).

Каждый MOF-файл содержит описание одного или (чаще) нескольких ***классов*** (***classes***), которые содержат:

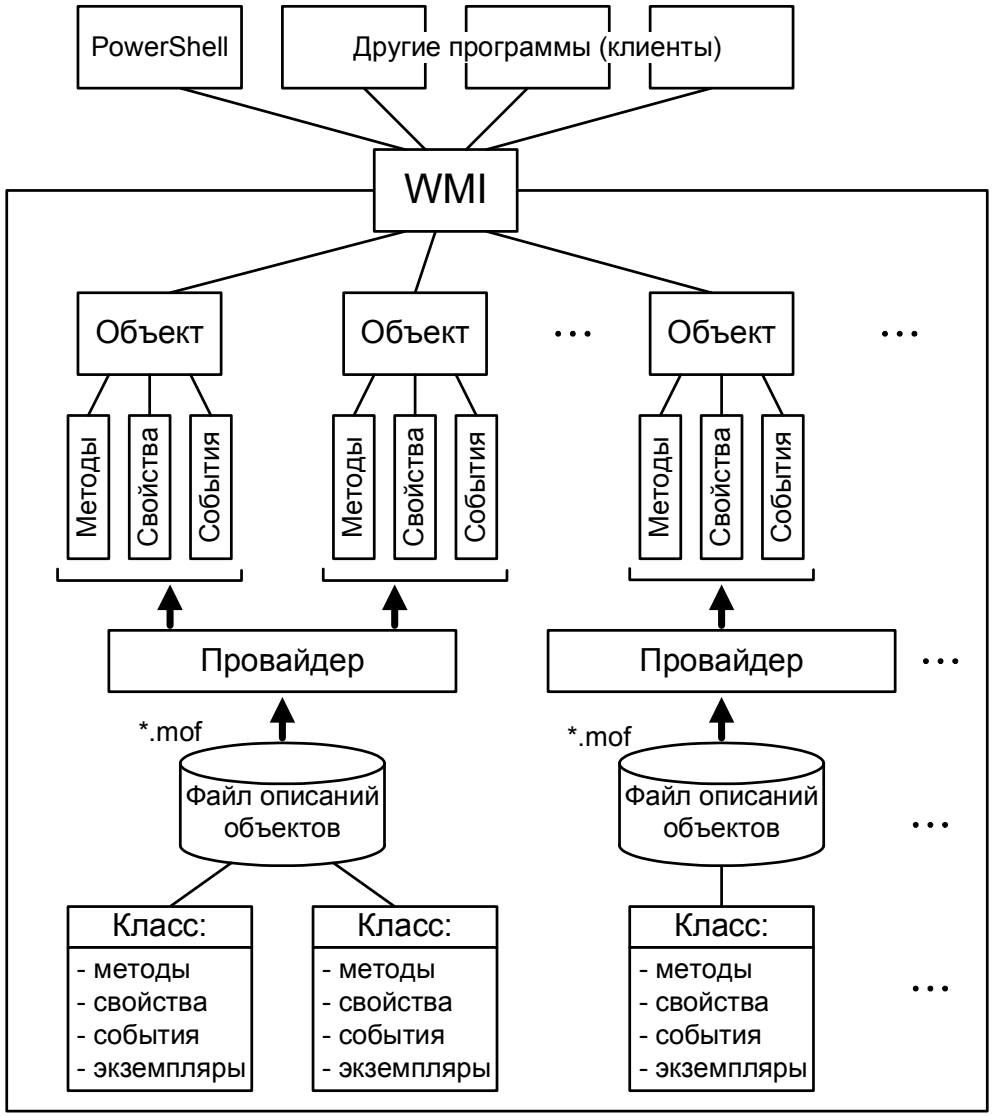
– ***методы*** (***methods***)

– ***свойства*** (***properties***)

– ***события*** (***events***)

– ***экземпляры*** классов (***instances***).

MOF-описания загружаются одним из ***провайдеров*** (***providers***), после чего их классы включаются в ***пространства имен*** (***namespaces***), которые представляют собой некоторое подобие иерархической файловой системы. Провайдеры предоставляются как системой, так и отдельными приложениями.



Программная модель WMI

В собственном скрипте PowerShell начинать обычно придется с команды:

```
get-WMIObject имя_объекта
```

например

```
get-WMIObject Win32_Timezone
```

Полученный объект пока ни для чего не используется, поэтому его содержимое просто выводится в консоль:

```
Bias SettingID                                  Caption
---- ---------                                  -------
 180                                           (UTC+03:00) Kaliningrad, Minsk
```

Это правило действует для большинства объектов, но не для всех; к тому же для многих классов вывод содержимого класса означает формально выполненную сериализацию, а не удобный для чтения формат.

Формат вывода можно задать самостоятельно, используя стандартные средства PowerShell:

```
Get-WmiObject \
  -Class Win32_UserAccount -Namespace root/cimv2 \
| Format-List Name,FullName
```

(Здесь дополнительно показаны опции: **`-Class`** и **`-Namespace`**. В данном случае их можно было бы опустить: они лишь явным образом подтверждают поведение «по умолчанию».)

Кроме ресурсов и функционала своей (локальной) системы, можно извлекать сведения из других машин в сети, для этого служит опция **-ComputerName** (имя «.» обозначает локальную систему). Например, можно попытаться запросить список процессов (как и в предыдущем примере, целесообразно воспользоваться преобразованием выводимых данных):

```
Get-WmiObject \
  -Class Win32_Process \
  -ComputerName Name_of_Remote_Computer
```

Дальнейшее продвижение зависит от владения набором объектов, их свойствами и методами. Полный список доступных объектов:

```
get-WMIObject -list
```

будет полезен, но не очень удобен из-за слишком большого размера. Например, в Windows 7 насчитывается почти 8 тысяч классов.

В версии PowerShell 3.0 введен параллельный набор команд и классов с префиксами «**cim**». В основном он дублирует имевшиеся ранее средства, но в некоторых случаях обеспечивает больше возможностей.

Проблемы при использовании WMI:

– нестабильность содержимого объектов (есть существенные отличия в зависимости от версии Windows) при недостаточно подробном и прозрачном их документировании;

– то же в отношении приложений: от версии к версии могут меняться содержимое классов, предоставляться или не предоставляться провайдеры, и т.п.;

– существующие универсальные, в первую очередь имеющие GUI, приложения для работы с объектами WMI часто бывают медленными и ресурсоемкими из-за того, что загружают и держат в памяти всю обширную иерархию WMI.

12.5 Протокол/служба управления сетями SNMP и распределенная база данных MIB

(Рассматривается лишь как пример организации управления распределенной системой.)

Simple Network Management Protocol (***SNMP***) – «протокол управления сетью», RFC 1157, 3416 и др.

Management Information Base (***MIB***) – база управляющей информации, RFC 1213, 1450, 1907 и др.

Архитектура «агент-менеджер». Невозможность централизованного хранения всех параметров системы приводит к использованию частичных ее моделей.

В упрощенном представлении:

MIB – хранение агентом сведений о своем узле в виде иерархии элементов данных унифицированного вида (а также менеджером – выбранного «среза» системы)

SNMP – запросы элементов данных

MIB 2, SNMP 2 – наличие не только информационных, но и управляющих элементов, модификация которых приводит к изменению режима работы узла.

Рис. – SNMP и MIB