

# Синхронизация ACL и структуры организации

## Часть 2

Вадим Андросов

**В статье описывается реализация основных функций, обеспечивающих поддержку списков контроля доступа и структуры организации в согласованном виде. Параллельно рассматриваются более сложные, но и более эффективные, методики оформления сценариев WSH.**

### Оформление сценариев

При разработке сложных надстроек на основе сценариев пришлось столкнуться с рядом неудобных моментов:

- повторное использование функций в разных сценариях с помощью «копирования-вставки» постоянно приводило к негативным последствиям – при изменениях подпрограмм не все файлы своевременно обновлялись;
- подобная проблема наблюдалась и с константами, которые нужно было заново объявлять в каждом файле сценария;
- сценарий временной подписки на события пригоден только в отладочных целях, реализовывать на его основе надстройку, широко использующую события, было нецелесообразно – с завершением работы сценария завершалось и слежение за событиями. Требовалась разработка дополнительных средств, поддерживающих жизнеспособность основного сценария, что существенно усложнило бы надстройку как с точки зрения разработки, так и сопровождения.

Это не просто неудобства – с ростом размера и сложности системы сценариев работать из-за них стало попросту невозможно. Поэтому я посчитал необходимым описать более удобную технологию. WSH поддерживает средства решения приведенных проблем.

Для начала рассмотрим повторное использование кода. Существует несколько уровней оформления сценариев.

На первом, самом простом, которым и пользуется большинство разработчиков, мы не имеем возможности создания собственных библиотек. Файл сценария не содержит ничего, кроме исходного кода программы на одном из языков (расширение зависит от используемого языка, например, для VBScript это vbs).

Следующий уровень – это Windows Script Files. Они имеют расширение wsf. По сути это xml-документ, содержащий код сценария. Будет приведено только описание, достаточное для понимания исходного кода статьи, для более подробного изучения этого вопроса можно обратиться к [8]. Этот уровень рассмот-

рим схематически на искусственных примерах, поскольку он понадобится только для перехода к следующему способу упаковки сценариев.

Допустим, у нас есть файл сценария, содержащий некоторые общие функции и константы (my\_library.vbs):

```
const SOME_CONST = 3

function add(a, b)
add = a + b
end function
```

Тогда wsf-сценарий, использующий данную библиотеку, может выглядеть так:

```
<?xml Version="1.0" ?>
<package>
<job>
<script language = "vbscript" _
src = "c:\scripts\ _
my_library.vbs" />
<script language = "vbscript">
<![CDATA[
msgbox add(1, SOME_CONST)
]]>
</script>
</job>
</package>
```

Первая строка – стандартное начало xml-документа. Она может быть опущена, однако в этом случае будет ис-

пользована менее строгая синтаксическая проверка файла, что чревато лишними ошибками.

Рассмотрим использованные теги:

- **package** – содержит весь код сценария.
- **job** – содержит одно задание. Теоретически файл сценария может содержать сразу несколько заданий. В этом случае требуется их именование с помощью свойства id:

```
<job id = "some_job">
```

Затем при запуске сценария нужно указывать, какое задание выполнить:

```
cscript my_script.wsf //Job:some_job
```

Если запустить файл с несколькими заданиями, будет выполнено только первое.

- **script** – содержит исходный код сценария или ссылку на внешний файл. Требуется указание языка сценария. Таким образом, существует теоретическая возможность использования сценариев на разных языках в одном файле Windows Script. Это имеет смысл, пожалуй, только при объединении в единое целое работ нескольких человек, которые не смогли договориться о едином языке программирования. Исходный текст сценариев рекомендуется помещать в блок CDATA. Внутри этого блока интерпретатор не будет искать xml-тегов, что позволит избежать сообщений об ошибках, когда в сценариях используются похожие на теги строки, например:

```
msgbox "I like <script> tag!"
```

Задание может содержать любое количество сценариев, выполняются они последовательно.

Такой подход уже позволяет эффективно повторно использовать код, однако содержит на первый взгляд несущественный, но на практике очень досадный недостаток. Для ссылок на файлы библиотек он требует указания полного пути к файлу. Исключением является случай, когда включаемый сценарий находится в том же каталоге, но этот вариант тоже не самый удобный. В результате получаем большое количество проблем при установке и сопровождении.

Проблему полностью решает оформление сценария в виде COM-объекта. Его можно зарегистрировать стандартным образом и затем использовать вне зависимости от физического расположения файла компонента. Для данной надстройки важно только это преимущество, однако объектно-ориентированные технологии (пусть несколько ограниченные – предлагается только полноценная инкапсуляция, что, однако, совсем не мало) позволяют существенно повысить качество работы.

Windows Script Component – это также xml-файл, содержащий исходный код сценария. Начнем с простого искусственного примера для описания основных позиций, а затем перейдем к непосредственной реализации

надстройки. Более подробное описание технологии можно найти в [8].

Создадим объект, служащий для суммирования чисел. Он будет содержать одно свойство – текущее значение (value) и два метода: add(число) – добавление к текущему значению заданного числа и reset() для обнуления текущего значения. При этом непосредственное изменение свойства value будет запрещено – пользователи объекта нашего класса смогут только получать его значение. Сначала приведем исходный код этого компонента. Писать заготовку кода нет необходимости – для этого существует бесплатное приложение «Windows Script Component Wizard», которое можно загрузить с сайта Microsoft. Естественно, существуют и коммерческие приложения с более широкими функциями, но для наших целей будет вполне достаточно этого мастера. Несмотря на то что заготовка может быть создана без особых усилий и вручную, я бы советовал пользоваться мастером, так как, кроме всего прочего, он генерирует идентификатор объекта (classid).

```
<?xml version="1.0"?>
<component>
<registration
  description="Accumulator"
  progid="DynamicGroup.Accumulator"
  version="1.00"
  classid="{b4e91e0a-967b-486b-8312-77366e6fd66c}"
>
</registration>
<public>
  <property name="value">
    <get/>
  </property>
  <method name="add">
    <PARAMETER name="addValue"/>
  </method>
  <method name="reset">
  </method>
</public>
<script language="VBScript">
<![CDATA[
dim value
value = 0
function get_value()
  get_value = value
end function
function add(addValue)
  value = value + addValue
  add = value
end function
function reset()
  value = 0
  reset = 0
end function
]]>
</script>
</component>
```

Итак, исходный код очень напоминает рассмотренные Windows Script Files. Рассмотрим только отличия. Во-первых, все описание компонента помещается в тело тега component.

Внутри содержится ряд тегов:

- **registration** – содержит информацию, используемую при регистрации компонента.
- **description** – краткое описание компонента.
- **progid** – имя, с использованием которого будет создаваться экземпляр объекта. То есть для создания этого объекта нужно будет выполнить команду:

```
set accumObj = Createobject("DynamicGroup.Accumulator")
```



Имя могло бы быть и просто Accumulator, однако лучше пользоваться составными, чтобы избежать совпадений с существующими объектами.

- **version** – версия компонента.
- **classid** – уникальный идентификатор класса. Для его генерации очень рекомендуется пользоваться специальными утилитами, одной из которых является Windows Script Component Wizard.
- **public** – интерфейс класса. Пользователи нашего класса будут иметь доступ только к сущностям, объявленным в этом разделе.
- **property** – свойство класса. Доступ к нему может быть получен следующим образом: `accumObj.value`. Поскольку внутри определен только `get`, свойство доступно только для чтения. Другими словами, `accumObj.value = 1` приведет к ошибке.
- **method** – объявляет метод класса с заданным именем. Если метод имеет параметры, они перечисляются в качестве вложенных тегов `PARAMETER`. Наш класс содержит два метода: `add` с одним параметром и `reset` без параметров.
- **script** – в этом разделе содержится непосредственная реализация интерфейса класса. Разработчик обязан только определить методы, необходимые для поддержки реализуемого интерфейса. Для чтения свойства мы должны определить метод с именем `get_<имя_свойства>`, в нашем примере – `get_value`. Если бы была допустима и запись свойства, понадобился бы метод `put_value`. Методы реализуются посредством определения функций с указанными именами. Также допускается включение любого количества вспомогательных функций и переменных. Естественно, пользователи COM-объекта доступа к ним не получают. Файл может содержать любое количество тегов `script`, а также использовать внешние файлы сценариев, как и в файлах `wsf`.

Файл сценария сохраняется с расширением `wsc` (Windows Script Component). После этого класс нужно зарегистрировать в операционной системе. Это можно сделать из проводника с помощью контекстного меню «Установить (Register)» (см. **рисунки**).

После выполнения установки компонент становится доступным для использования.

Рассмотрим пример сценария использования нашего класса. Программа выводит начальное значение счетчи-

ка, затем добавляет к нему два числа и, наконец, показывает результат.

```
set accumObj = CreateObject("DynamicGroup.Accumulator")
Msgbox accumObj.Value
accumObj.Add(3)
accumObj.Add(4)
Msgbox accumObj.Value
```

Мы получаем возможность использовать зарегистрированные компоненты на основе только их имени, что гораздо удобнее ссылок на основе полных или даже относительных путей к файлам сценариев.

С помощью контекстного меню «Отключить» (см. **рисунки**) можно отменить регистрацию компонента. При изменении интерфейса или места расположения класса его нужно сначала отключить, а затем заново зарегистрировать. Если же меняется только реализация класса с сохранением интерфейса, то повторная регистрация не требуется. Когда создается много классов, гораздо удобнее регистрировать и удалять их в пакетном режиме, для этого следует воспользоваться следующими командами:

- **Regsvr32 Accumulator.wsc** – для регистрации компонента;
- **Regsvr32 /u Accumulator.wsc** – для его удаления.

Итак, мы рассмотрели основные положения упаковки сценариев. Теперь можно переходить к непосредственной реализации надстройки.

## Основные функции

Рассмотрим базовые функции, которые будут использоваться при реализации реакции на события. Функция `getOU` получает путь к родительской организационной единице заданного объекта. Подобная функция использовалась в [1].

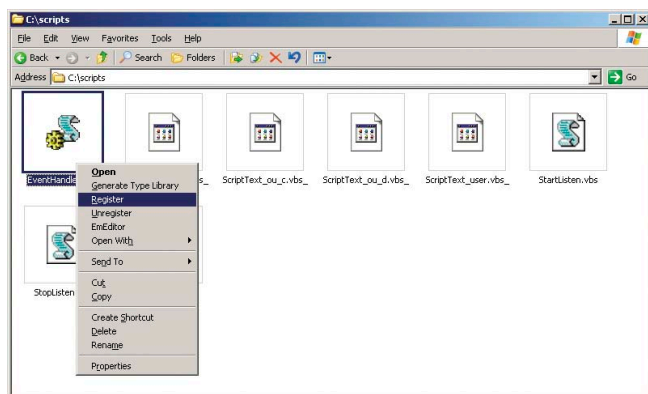
```
function getOU(dn)
    getOU = "LDAP://" & right(dn, len(dn) - instr(1, dn, "\", vbTextCompare))
end function
```

Также нам понадобится другая функция из [1] – `getParent`, которая имеет похожую функциональность, но возвращает пустую строку, когда достигнут корень иерархии (первый доменный элемент).

```
function getParent(dn)
    if UCase(Left(dn, 2))="DC" then
        getParent = ""
    else
        getParent = getOU(dn)
    end if
end function
```

Функция `isBadLink` проверяет, указывает ли данная ссылка (GUID) на существующий объект. В качестве параметра она получает идентификатор объекта. Функция делает попытку привязки к объекту на основе его GUID и в случае ошибки возвращает истину (т.е. да, ссылка плохая).

```
Function isBadLink(aLink)
    on Error Resume Next
    GetObject("LDAP://" & aLink & ">")
    isBadLink = Err > 0
    On Error Goto 0
end Function
```



Регистрация компонента

В дальнейшем эта подпрограмма используется для определения корректности ссылок на привязанные к организационным единицам группы (для каждого типа существует отдельная функция проверки).

```
function isBadSingle(ou)
isBadSingle = isBadLink(ou.groupSingle)
end function

function isBadTree(ou)
isBadTree = isBadLink(ou.groupTree)
end function
```

Следующая функция добавляет пользователя на основе его отличительного имени (DN) в заданную группу. Если целевая группа уже содержит пользователя, функция завершает работу.

```
function addToAttached(groupObj)
if groupObj.isMember(dn) then exit function
groupObj.add(dn)
groupObj.setInfo
end function
```

Здесь и далее dn указывает на отличительное имя объекта, с которым связано обрабатываемое событие. Это будет свойство конечного класса обработки событий, в который мы и включим все эти функции. В текущем случае речь идет о пользователях. Код непосредственной обработки событий для разных классов во многом похож.

```
set handler = createObject("DynamicGroup.EventHandler")
handler.dn = targetEvent.TargetInstance.ADSPATH
handler.on...
```

Это пример обработки событий манипуляций с объектами пользователей, однако первые две строчки остаются неизменными для всех событий. Сначала создается объект-обработчик событий (о подробностях его реализации – позже), затем инициализируется отличительное имя источника события (именно оно используется внутри функций под именем dn). В конце вызывается необходимый для каждого события метод.

Потребуется и подпрограмма, удаляющая пользователя из группы. Здесь все несколько сложнее. Дело в том, что эта операция проводится, когда из контейнера удаляется пользователь (переносится в другую организационную единицу, простое удаление пользователя и так приводит к его исчезновению из всех групп).

Например, это происходит, когда пользователь перемещается из одного контейнера в другой. При этом он должен быть отключен от группы типа Single контейнера-источника и включен в такую группу получателя. Перемещение сопровождается последовательным возникновением событий удаления из одной организационной единицы и создания в другой.

Однако при обработке события удаления можно получить только «устаревшее» отличительное имя источника, с помощью которого больше нельзя привязаться к объекту. Поэтому потребуется искать пользователя в группе на основе общего имени (Common Name) и, находя, удалять. Для выделения общего имени (которое не меняется при перемещении) из отличительного используется функция extractCN:

```
function extractCN()
pos = instr(1, UCase(dn), "CN=", vbTextCompare)
if pos = 0 then
extractCN = ""
exit Function
end if
extractCN = right(dn, len(dn) - pos - 3 + 1)
pos = instr(1, extractCN, ",", vbTextCompare)
extractCN = Left(extractCN, pos - 1)
end Function
```

Подчеркнем, что возможности привязаться к объекту и получить его общее имя с помощью свойства obj.cn нет, поскольку dn содержит путь к удаленному объекту. Следующая функция позволяет найти в группе пользователя по имени и получить его новое актуальное отличительное имя.

```
function userDNInGroup(groupObj)
userDNInGroup = ""
cn = extractCN
for each userObj in groupObj.members
if userObj.cn = CN then
userDNInGroup = userObj.ADSPATH
exit function
end if
next
end function
```

Подпрограмма сравнивает имена всех членов заданной группы с целевым общим именем и при совпадении возвращает полный путь к найденному объекту. В случае неудачного поиска возвращается пустая строка. Тогда функция удаления перемещенного пользователя из привязанной группы будет выглядеть так:

```
function removeFromAttached(groupObj)
newDN = userDNInGroup(groupObj)
if isEmpty(newDN) then exit function
groupObj.remove(newDN)
groupObj.setInfo
end function
```

Следующая функция будет применяться при отключении контейнера от родительского, т.е. для поддержки операций перемещения поддеревьев. Применяемый способ представления иерархических структур с помощью паттерна «Компоновщик» [3] позволяет свести операцию отключения поддерева к удалению привязанной к контейнеру группы типа Tree из такой группы родительского отдела. В качестве первого параметра функция получает идентификатор группы типа Tree дочернего отдела, второго – объект родительской организационной единицы.

```
function unRegisterInParent(treeGroupGUID, parentOU)
if not isGroupExist(parentOU.groupTree) then exit function
set treeGroup = getObject("LDAP://<GUID=" & _
parentOU.groupTree & ">")
set ou2Delete = getObject("LDAP://<GUID=" & _
treeGroupGUID & ">")
treeGroup.remove(ou2Delete.ADSPATH)
treeGroup.setInfo
end function
```

Вся работа функции сводится к привязке к объектам групп и удалению из группы tree родителя такой группы дочернего подразделения.

Ну и последняя вспомогательная функция будет использоваться в случаях, когда удаляется группа типа tree подразделения. В этом случае ее функции требуется возложить на такую группу родительского подразделения.

```
Function unattachTreeGroup(ou)
    parentPath = getParent(ou.distinguishedName)
    if not isEmpty(parentPath) Then
        set parentOU = getObject(parentPath)
        if not isEmpty(parentOU.groupTree) then
            ou.groupTree = parentOU.groupTree
            ou.setInfo
            set parentGroup = getObject("LDAP://<GUID=" & _
                parentOU.groupTree & ">")
            attach parentGroup, ou
        end if
    else
        ou.putEx ADS_PROPERTY_CLEAR, "groupTree", ""
        ou.setInfo
    end if
end Function
```

Подпрограмма проверяет, имеет ли родитель присоединенную группу типа tree. Если да, то эта группа подключается к текущей организационной единице. В противном случае ссылка на группу поддержки поддеревьев просто обнуляется.

Теперь можно перейти к непосредственной реализации реакции на интересующие нас события.

## Функции реакции на события

Как уже упоминалось выше, код обработки всех событий состоит из 3 строк, первые две из которых всегда одинаковы.

```
set handler = createObject("DynamicGroup.EventHandler")
handler.dn = targetEvent.TargetInstance.ADSIPath
```

Третья строка специфична для каждого вида события. Рассмотрим все варианты.

События, связанные с операциями над объектами типа user. Нас интересуют события создания и удаления этих объектов. Обработаться они будут посредством вызова следующего метода:

```
handler.onUserEvent(targetEvent.path_.class)
```

Метод будет вызываться в ответ на два типа событий, для каждого из них в программе были определены специальные константы (события создания и удаления пользователей):

```
const EVENT_CREATE_INSTANCE = "__InstanceCreationEvent"
const EVENT_DELETE_INSTANCE = "__InstanceDeletionEvent"
```

Рассмотрим реализацию обработчика:

```
function onUserEvent(eventType)
    set ouObj = getObject(getOU(dn))
    if isGroupExist(ouObj.groupSingle) then
        set grSingle = getObject("LDAP://<GUID=" & _
            ouObj.groupSingle & ">")
        if eventType = EVENT_CREATE_INSTANCE then
            addToAttached grSingle
        elseif eventType = EVENT_DELETE_INSTANCE then
            removeFromAttached grSingle
        end if
    end if
end function
```

Метод привязывается к объекту группы типа Single текущего контейнера и в зависимости от типа события добавляет в нее пользователя или удаляет его.

Из операций над группами нас интересует только удаление, чтобы мы могли при необходимости отключить удаленные объекты от контейнеров и избежать появления «повис-

ших» ссылок, которые ни на что не указывают. Последняя строчка кода обработки события будет иметь вид:

```
handler.onGroupDelete()
```

Рассмотрим реализацию этого метода:

```
function onGroupDelete()
    set ou = getObject(getOU(dn))
    if isBadSingle(ou) then
        ou.putEx ADS_PROPERTY_CLEAR, "groupSingle", ""
        ou.setInfo
    end if
    if isBadTree(ou) then
        unattachTreeGroup ou
    end if
end function
```

Функция подключается к контейнеру, в котором находилась удаленная группа, и проверяет ссылки на присоединенные группы (типа Tree и Single). Если некоторые ссылки (GUID) больше не указывают на существующие объекты, то они корректным образом затираются.

События создания и удаления организационных единиц, в отличие от пользователей, обрабатываются по отдельности. В ответ на создание подразделения выполняется сценарий обработки события со следующей третьей строчкой:

```
handler.onOuCreate()
```

Рассмотрим реализацию метода:

```
function onOuCreate()
    set ouObj = getObject(dn)
    if not isGroupExist(ouObj.groupTree) then exit function
    parentPath = getParent(dn)
    if isEmpty(parentPath) then exit function
    set parentOU = getObject(parentPath)
    registerInParent ouObj, parentOU
end function
```

При создании объекта организационной единицы, имеющей прикрепленную группу типа Tree, он должен быть корректно зарегистрирован в родительском подразделении. Такая реакция на событие создания позволит корректно обрабатывать перемещение поддеревьев структуры организации, поскольку, напомним, перемещение объекта сопровождается событиями его удаления из старого контейнера и создания в новом. Удаление также требуется обработать для поддержки корректности модели. При удалении выполняется сценарий обработки со следующей последней строчкой:

```
handler.onOuDelete(targetEvent.TargetInstance.DS_groupTree)
```

В процедуру обработки передается дополнительный параметр – указатель на прикрепленную к контейнеру группу типа Tree. Дело в том, что здесь снова производится обработка события удаления, когда сценарий располагает отличным именем уже несуществующего объекта организационной единицы. Для ее поиска по имени, наподобие того, как это было реализовано с пользователями, потребовался бы уже перебор всей иерархии, где могут встретиться объекты с совпадающими названиями. Кроме того, здесь нам не требуется собственно объект организационной единицы – достаточно присоединенной груп-

пы типа Tree, корректную ссылку на которую можно извлечь и из устаревшего объекта (в отличие от отличительного имени, GUID группы продолжает указывать на корректный объект). Обратите внимание, что присоединенная группа – наше расширение схемы Active Directory для объектов класса «organizationalUnit». К счастью, пользовательские расширения отражаются и на связанных классах ds\_organizationalUnit, только здесь новые свойства появляются с тем же именем плюс приставка DS. То есть если создавалось свойство с именем groupTree, то в классе organizationalUnit оно появится под именем DS\_groupTree. Значение этого свойства будет передано в функцию обработки события удаления организационной единицы.

```
function onOuDelete(treeGroupGUID)
    if not isGroupExist(treeGroupGUID) then exit function
    parentPath = getParent(dn)
    if isEmpty(parentPath) then exit function
    set parentOU = getObject(parentPath)
    unRegisterInParent treeGroupGUID, parentOU
end function
```

В приведенном методе проверяется наличие присоединенной группы типа Tree, определяется родительская организационная единица по отношению к удаляемой и вызывается уже рассмотренная выше функция отключения от родительского контейнера unRegisterInParent.

Поместим рассмотренные методы реакции надстройки на события в один класс DynamicGroup.EventHandler. Возможно, логичнее было бы сделать отдельный класс для каждого вида события, однако для упрощения изложения остановимся на одном. Итак, класс должен содержать по одному методу для событий каждого типа. В результате наш класс должен содержать четыре метода:

- **onOuCreate(eventType)** – обработчик события создания организационной единицы;
- **onOuDelete(treeGroupGUID)** – обработчик события удаления организационной единицы;
- **onGroupDelete()** – обработка удаления групп;
- **onUserEvent(eventType)** – обработчик событий над пользователями.

Кроме того, класс обработчика должен быть предварительно проинициализирован отличительным именем (Distinguished Name) объекта, действие над которым и спровоцировало то или иное событие. Для этой цели введем свойство DN, общее для всех типов событий. Логика работы всех методов уже рассмотрена выше, поэтому приведем лишь каркас класса-обработчика всех событий:

```
<?xml version="1.0"?>
<component>
<registration
    description="EventHandler"
    progid="DynamicGroup.EventHandler"
    version="1.0.0"
    classid="{d55f71f5-9373-4f13-af5a-57248b3ade00}"
>
</registration>
<public>
    <property name="dn">
        <get/>
        <put/>
    </property>
    <method name="onOuCreate">
    </method>
```

```
<method name="onOuDelete">
    <PARAMETER name="treeGroupGUID"/>
</method>
<method name="onGroupDelete">
</method>
<method name="onUserEvent">
    <PARAMETER name="eventType"/>
</method>
</public>
<script language="VBScript">
<![CDATA[
dim dn

function get_dn()
    get_dn = dn
end function


function put_dn(newValue)
    dn = newValue
end function

function onOuCreate()
    '...
end function

function onOuDelete(treeGroupGUID)
    '...
end function

function onGroupDelete()
    '...
end function

function onUserEvent(eventType)
    '...
end function
'Реализация вспомогательных функций ...
]]>
</script>
</component>
```

На этом завершается разработка основной логики надстройки. Остается окончательно заставить все это работать. В следующий раз будут рассмотрены методы установки обработчиков необходимых событий. В отличие от [1], в этой работе будут использоваться постоянные обработчики событий как более надежные. Для поддержки их функционирования не требуется постоянно запущенного сценария, кроме того, они устойчивы к перезагрузкам системы. 

1. Андросов В. Реализуем нестандартные правила управления доступом на основе архитектуры организации в Windows Server 2003. //Системный администратор, №10, 2007 г. – С. 48-58.
2. Сайт Microsoft – [www.microsoft.com](http://www.microsoft.com).
3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. «Приемы объектно-ориентированного проектирования. Паттерны проектирования». – СПб.: Питер, 2007. – 366 с.
4. Чарли Рассел, Шарон Кроуфорд, Джейсон Джеренд. «Windows server 2003 +SP1 и R2. Справочник администратора». – М.: Издательство «ЭКОМ», 2006. – 1424 с.
5. Alain Lissior. «Understanding WMI Scripting». – Digital Press, 2003.
6. Microsoft Development Network – [msdn.microsoft.com](http://msdn.microsoft.com).
7. Чекмаев А.Н. «Windows 2000 Active Directory». – СПб.: БХВ-Петербург, 2001. – 400 с.
8. Don Jones, Jefferey Hicks «Advanced VBScript for Microsoft Windows Administrators». – Washington: Microsoft Press, 2006.
9. Microsoft® Windows® 2000 Scripting Guide «Enhanced WMI Monitoring Scripts».
10. Андросов В. Синхронизация ACL и структуры организации. Часть 1. //Системный администратор, №12, 2007 г. – С. 36-41.