

phpGACL - русская документация

Содержание

Содержание

О скрипте

- Что это?

- Где взять?

- Что нужно для запуска?

- Авторы

Введение

- Понимание контроля действия

- Управление контролем доступа при помощи phpGACL

- Подробный контроль доступа

- Многоуровневые группы

- Как phpGACL определяет разрешения?

- Добавление группы Добавление людей

- Устранение конфликтов

- Обозначение объектов доступа

- Добавление разделов

- Множество целей

- Объекты расширения доступа (AXO)

Инсталляция

- Базовая настройка

- Дополнительная настройка

Использование phpGACL в вашей программе

- Базовое использование

- Профессиональное использование

 - Использование административного интерфейса

 - Описание API

 - ACL

 - Groups (Группы)

 - Access Objects (ACO, ARO, AXO) (Объекты доступа)

 - Access Objects Section (разделы объектов доступа)

Mike Benoit (ipso@snappymail.ca)
James Russell (james-phpgacl@ps2-pro.com)
Karsten Dambekalns (k.dambekalns@fishfarm.de)
Русский перевод Феськов Кузьма (kuzma@russofile.ru) (<http://php.russofile.ru>)
Copyright © 2002,2003, Mike Benoit
Copyright © 2003, James Russell
Copyright © 2003, Karsten Dambekalns
Document Version: 672
Last Updated: 5/20/03 - 18:55:08

phpGACL

Что это?

PhpGACL – это набор функций, который позволяет определить права доступа к произвольным объектам (например, страницам, базам данных и так далее) другим объектам (например, пользователями, удаленными хостами и так далее).

Это предполагает подробный контроль доступа с простым управлением и высокой скоростью.

Скрипт написан на PHP (отсюда его название – phpGACL) – это популярный скриптовый язык, который обычно используется для динамического создания страниц сети. GACL часть phpGACL – Родовой список контроля доступа.

Где взять?

PhpGACL хостится на sourceforge.net здесь <http://phpGACL.sourceforge.net/>

Что нужно для запуска?

PhpGACL требует реляционную базу данных для хранения информации о правах. Доступ к базе данных осуществляется через абстрактный класс доступа к базам данных ADOdb (<http://php.weblogs.com/adodb>). Данный класс позволяет работать с такими базами данных как: PostgreSQL, MySQL, Oracle.

Поскольку phpGACL написан на языке PHP, то он требует установки PHP версии не ниже 4.2. ACL (Администрирование списка контроля доступа) дополнена веб-интерфейсом, потому необходимо наличие веб-сервера с поддержкой PHP, например, Apache (<http://httpd.apache.org/>).

Авторы

Mike Benoit (ipso@snappymail.ca) – автор и менеджер проекта.
James Russel (james-phpgacl@ps2-pro.com) и

Karsten Dambekalns (k.dambekalns@fishfarm.de) – авторы документации.
Feskov Kuzma (kuzma@russofile.ru) – русский перевод
Последняя версия перевода всегда здесь: <http://php.russofile.ru>

Введение

Понимание контроля действия

Хан – капитан “Сокола тысячелетия” и Чуви – его второй офицер. Они взяли на борт несколько пассажиров: Люка, Оби-Вана, R2-D2 и СЗРО. Хан должен определить ограничения доступа для различных мест (комнат) корабля, например: кабина, машинный зал, двигатели и внешнее оружие.

Хан сказал: “Я и Чуви могут иметь доступ всюду, но, после особенно грязного ремонта гипердвигателя, я запрещаю Чуви когда-либо подходить к машинному залу. Пассажиры ограничены пассажирским залом”.

С этого момента, давайте будем считать, что доступ является Boolean (Булевым), то есть, результат запроса на доступ персонажа к какому-либо помещению является “Разрешить” или “Не разрешить”, и нет никакого среднего результата.

Если мы нанесем это утверждение на матрицу доступа, показывающую, кто имеет доступ куда-либо, то это бы выглядело примерно так (О – доступ разрешен, X – доступ запрещен):

Кто / Куда	Кабина	Пассажирский зал	Оружие	Машинный зал
Хан	О	О	О	О
Чуви	О	О	О	Х
Оби-Ван	Х	О	Х	Х
Люк	Х	О	Х	Х
R2-D2	Х	О	Х	Х
СЗРО	Х	О	Х	Х

Колонки показывают список комнат, к которым Хан может ограничить доступ, а ряды показывают персон, которые могли бы запросить доступ к тем или иным комнатам. Более широкое объяснение:

“комнаты” - это вещи (объекты) для управления доступа к ним. Мы называем эти объекты **ACO** (*Access Control Objects* - Объекты контроля доступа);

“люди” - “вещи (объекты), запрашивающие доступ”. Мы называем их **ARO** (*Access Request Objects* - Объекты запроса доступа). Люди запрашивают доступ к комнатам, или, в нашей терминологии, AROs запрашивают доступ к ACOs.

Так же есть третий тип объекта – **AXO** (*Access eXtension Objects* – Объекты расширения

доступа), но о нем мы поговорим позже. Эти объекты имеют много атрибутов и упоминаются вместе как Объекты доступа.

Управление доступом, используя матрицу доступа, подобной выше, имеет преимущества и недостатки:

Плюсы:

- очень подробный – есть возможность управления доступом на уровне конкретного человека, если есть такая необходимость;
- легко увидеть, кто имеет доступ и к чему. Ответ находится в пересечении человека и комнаты.

Минусы:

- трудно управлять на крупной матрице доступа. 6 пассажиров и 4 места – это просто, но что, если у вас тысячи пассажиров и сотни мест, и вы должны ограничить доступ большими группами и сразу, но сохранить при этом подробный контроль для отдельного индивидуума? Это означало бы большое и трудоемкое регулирование матрицы, а так же поставило бы очень трудную задачу проверки матрицы на правильность;
- трудно суммировать и визуализировать. Предыдущий пример довольно прост, его можно записать в несколько предложений (смотрите высказывание Хана), но что, если матрица напоминает такую?

Кто / Куда	Кабина	Пассажирский зал	Оружие	Машинный зал
Хан	0	0	0	0
Чуви	0	X	0	X
Оби-Ван	X	0	X	X
Люк	0	0	0	X
R2-D2	X	0	X	0
СЗРО	0	0	X	0

Эта матрица не настолько очевидна для суммирования, и не понятна для читателя, почему те или иные решения были приняты.

Управление контролем доступа при помощи phpGACL

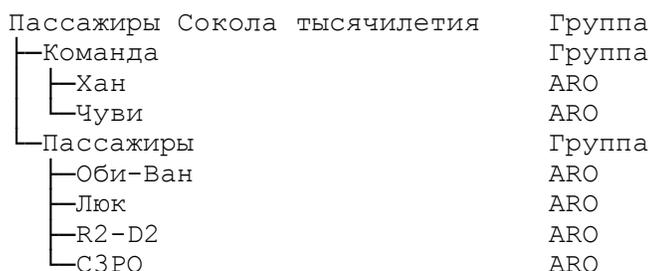
Ясно, что для контроля над большими системами и сложными ситуациями этот подход “матрицы доступа” является неподходящим. Мы нуждаемся в лучшей системе, которая поддерживает плюсы (подробный контроль и понятную визуализацию того, кто имеет доступ и к чему) и удаляет неудобства (трудность суммирования, трудность управления большими группами людей сразу). Одно решение – phpGACL.

PhpGACL не описывает доступ от “восходящего” подобно матрице доступа выше. Вместо

этого, описывает доступ сверху вниз, подобно текстовому описанию политики доступа Хана. Это – очень гибкая система, которая позволяет вам управлять доступом в больших группах, так же позволяет аккуратно суммировать политику доступа и легко видеть, кто имеет право на доступ и к чему.

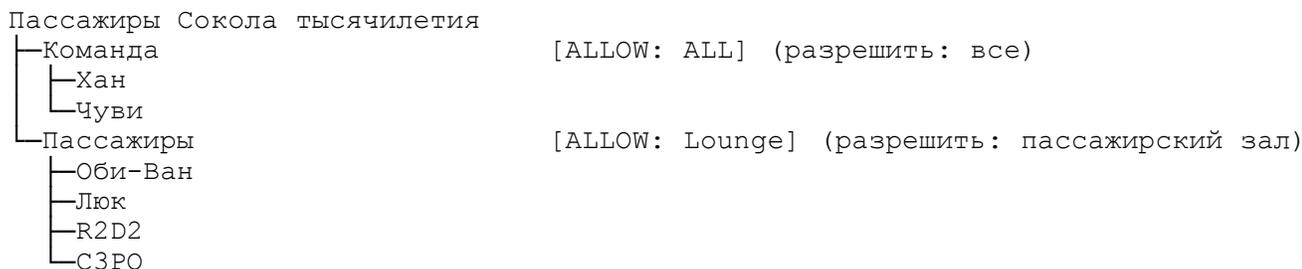
ARO-дерево определяет иерархию групп AROs (объектов запроса доступа). Это подобно представлению дерева папок и файлов. “Папки” - группы, “файлы” - AROs.

Давайте попробуем составить ACL-дерево для людей на судне Хана. Сначала мы определим некоторые категории для людей. Ясно, что Хан и Чуви управляют судном, а остальные объекты – это только пассажиры.



Это дерево не определяет никакой политики, оно просто показывает, каким образом мы группируем людей, которые могли бы запросить доступ (AROs).

Мы задаем ограничения доступа, создавая инструкции для определенной комнаты (ACO) какой-то группе или AROs в дереве. Хан говорит: “По умолчанию никому нельзя назначить доступ ни к одной из комнат на Соколе тысячелетия. Но команда должна иметь доступ к каждой комнате. Пассажиры должны иметь доступ только к пассажирскому залу”.



Чтобы интерпретировать наше ARO-дерево, мы начинаем с вершины и двигаемся вниз.

Во-первых, политика по умолчанию состоит в том, чтобы запретить доступ ко всему. Ограничения были изменены только для команды, так что они теперь имеют доступ ко всему (ALL – синоним для всех комнат (кабина, машинный зал, пассажирский зал и оружие). Пассажиры имеют только доступ к пассажирскому залу.

Такой способ описывать политику доступа более ясен чем матрица доступа. Вы можете легко видеть кто имеет доступ к чему, а так же более легко определить, почему они имеют этот доступ (кажется очевидным, что Хан и Чуви имеют доступ ко всему, так как они сгруппированы в “Команду”).

Суммируем:

- ACO (объекты контроля доступа) – вещи, доступом к которым мы хотим управлять (например страницы сети, базы данных, комнаты и так далее);
- ARO (объекты запроса доступа) – вещи, которые запрашивают доступ (например, люди, отдаленные компьютеры и так далее);
- ARO-деревья – определяют иерархию **Групп** и AROs. Группы могут содержать другие группы и AROs;
- политика по умолчанию для дерева – запретить все (DENY: ALL);
- чтобы определить политику доступа, двигайтесь вниз по дереву, явно назначая разрешения Группам и AROs для каждого ACO, если в этом возникает потребность.

Подробный контроль доступа

Ой, мы кажется забыли про Чуви! Отнеся его к команде Хан косвенно дал ему доступ в машинный зал! Но, если помните, он не хотел этого делать. Он не хочет этого после того, как Чуви недавно ремонтировал гипердвигатель. Хан добавляет правило запрещения для Чуви:

Пассажиры Сокола тысячелетия

—Команда	[ALLOW: ALL] (разрешить: все)
—Хан	
—Чуви	[DENY: Engines] (запретить: машинный зал)
—Пассажиры	[ALLOW: Lounge] (разрешить: пассажирский зал)
—Оби-Ван	
—Люк	
—R2D2	
—C3PO	

Это пример того, как вы можете управлять политикой доступа в подробной манере. Нет необходимости перемещать Чуви в другую группу, мы просто запрещаем ему на более низком уровне.

Другой пример подробного контроля может возникнуть, когда на корабль напала Империя. Хан должен позволить человеку Люку использовать оружие и позволить роботу R2D2 восстановить гипердвигатель в машинном зале. Он может сделать это расширив их полномочия, которые определены им общим статусом группы “Пассажир”:

Пассажиры Сокола тысячелетия

—Команда	[ALLOW: ALL] (разрешить: все)
—Хан	
—Чуви	[DENY: Engines] (запретить: машинный зал)
—Пассажиры	[ALLOW: Lounge] (разрешить: пассажирский зал)
—Оби-Ван	
—Люк	[ALLOW: Guns] (разрешить: оружие)

```
├─R2D2 [ALLOW: Engines] (разрешить: машинный зал)
└─C3PO
```

Многоуровневые группы

Группы могут быть расширены на любой уровень в ARO-дереве. Например, мы могли бы добавить группу “Джедаи” к “Пассажирам”. Большинство пассажиров было бы категоризировано при “Пассажирах”, но Люк и Оби-Ван будут выделены в “Джедаи” и поэтому могут иметь некоторые привелегии (например, доступ в кабину):

```
Пассажиры Сокола тысячелетия
├─Команда [ALLOW: ALL] (разрешить: все)
│   └─Хан
│   └─Чуви
├─Пассажиры [ALLOW: Lounge] (разрешить: пассажирский зал)
│   └─Джедаи [ALLOW: Cockpit] (разрешить: кабину)
│       └─Оби-Ван
│       └─Люк [ALLOW: Guns] (разрешить: оружие)
├─R2D2 [ALLOW: Engines] (разрешить: машинное отделение)
└─C3PO
```

Как phpGACL определяет разрешения?

Когда компьютер судна (с запущенным phpGACL конечно же) проверяет доступ, единственный вопрос, который он может задать себе - “имеет ли человек X доступ к месту Y?” На языке phpGACL это будет выглядеть так: “Может ли ARO X иметь доступ к ACO Y?”

phpGACL определяет, имеет ли определенный человек доступ к определенному месту следуя от вершины дерева к указанному человеку, отмечая по пути явный контроль доступа для того места. Когда мы достигаем нужного человека, получаем конкретный уровень доступа который и является конечным результатом для возвращения. Таким образом мы осуществляем контроль доступа для групп, людей, но имеем возможность расширять эти уровни если нужно.

Пример 1. Мы спрашиваем: Имеет ли Люк доступ к пассажирскому залу?”

- устанавливаем правило по умолчанию DENY (запретить);
- движемся по дереву к Люку: *Пассажиры Сокола тысячелетия* -> *Пассажиры* -> *Джедаи* -> *Люк*;
- начинаем с вершины дерева и движемся к Люку: “узел” “Пассажиры Сокола тысячелетия” не говорит ни ничего не об одном из ACO. Оставляем DENY (запрет);
- идем дальше, “Пассажиры” - явно говорит, что “Пассажиры” имеют доступ к “пассажирскому залу”. Меняем внутренний результат на “ALLOW” (разрешить);
- движемся в узел “Джедаи” - он не упоминает “пассажирского зала” вообще – оставляем предыдущий результат;
- движемся в узел “Люк” - снова нет ничего о “пассажирском зале” - оставляем предыдущий результат;
- идти дальше некуда – останавливаемся. Возвращаемый результат будет “ALLOW” (разрешить).

Пример 2. Мы спрашиваем: “Имеет ли Чуви доступ в машинный зал?”

- устанавливаем правило по умолчанию DENY (запретить);
- движемся по дереву к Чуви: *Пассажиры Сокола тысячелетия* -> *Команда* -> *Чуви*;
- начинаем с вершины дерева и движемся к Чуви: “узел” “Пассажиры Сокола тысячелетия” не говорит ни ничего не об одном из АСО. Оставляем DENY (запрет);
- идем дальше в узел “Команда” - явно говорит нам - “Команда” имеет доступ к “машинному залу” - меняем внутренний результат на “ALLOW” (разрешить);
- движемся в узел “Чуви” - есть явное запрещение, которое говорит нам, что доступ в “машинный зал” закрыт. Меняем внутренний результат на “DENY” (запретить);
- идти дальше некуда, возвращаем результат “DENY” (запретить).

Вы можете видеть, что если группа явно не определяет никаких прав, то она наследует их от родительской группы (предыдущей). Если узел корня (“Пассажиры Сокола тысячелетия”) не определяет никаких разрешений, то он наследует разрешения по умолчанию. В данном случае “DENY ALL” (запретить все).

Это подразумевает несколько интересных моментов об ARO-дереве:

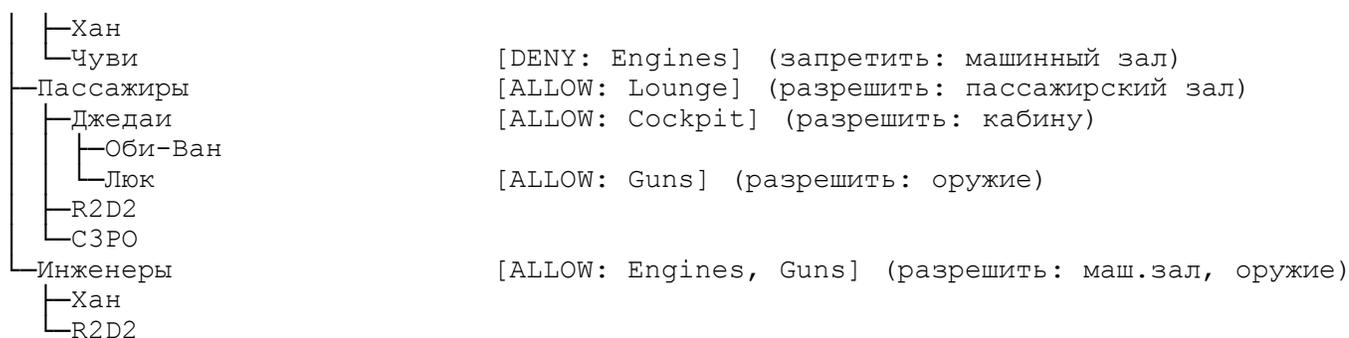
- ARO-дерево всегда показывает полный список AROs. Поэтому не имеет смысла спрашивать, имеет ли доступ к кабине Джабба, поскольку он не был определен в системе. Однако, phpGACL не проверяет существует ли ARO или АСО перед выполнением проверки. Так, если бы такой вопрос действительно имел бы место, ответ был бы “DENY” (запретить);
- ARO-дерево может не показывать некоторые определения АСО если они специально не определены в ARO-дереве. Например, Хан определил АСО “ванная комната”. Любопытно, подобный этому: “имеет ли Люк доступ в ванную комнату?”, даст отрицательный ответ “DENY” (запрещено), поскольку нигде в ARO-дереве “ванная комната” не упоминается. Имейте в виду, при исследовании ARO-деревя, некоторые АСО могут быть не видимыми.

Обратите внимание: при выяснении через phpGACL вопросов о доступе невозможно использовать Группы как AROs (даже при том, что это может казаться правильным). Например, невозможно ответить на вопрос “Какие пассажиры имеют доступ к оружию?” Полный ответ не Boolean (не Булевый) – DENY / ALLOW (отрицаю / позволяю), а более сложный “Люк и Оби-Ван могут, а R2D2 и СЗРО нет”. PhpGACL не предназначен для такого вида ответов.

Добавление групп

Хан видит, что ACL начинает выглядеть очень сложно. Есть очень много исключений. Возможно, следует сделать по-другому, например, создать новую группу “Инженеры”, которая будет содержать людей, которым позволен доступ в машинное отделение и оружию. Эта группа должна содержать Хана и R2D2, так как они могут восстановить двигатель и оружие. Это позволяет Хану удалить некоторые из “грязных” правил-исключений, это дает определенную выгоду в том, что дает ясное описание:

По умолчанию:	[DENY: ALL] (запретить: все)
Пассажиры Сокола тысячелетия	
└─Команда	[ALLOW: ALL] (разрешить: все)

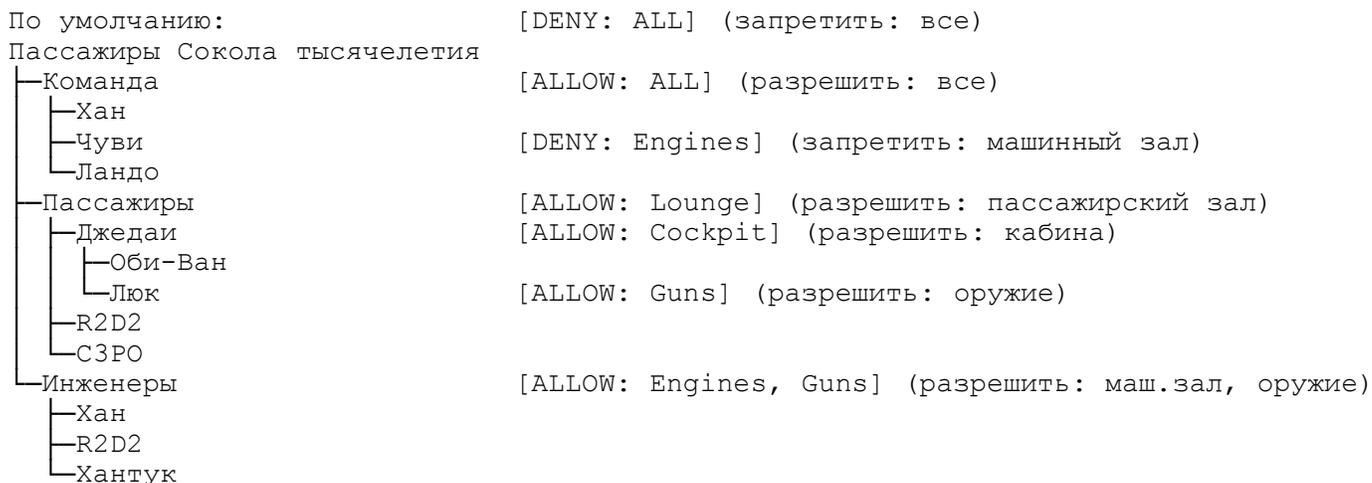


Вы можете прочитать это следующим образом: по умолчанию никто никуда не имеет право заходить, команда имеет доступ всюду (кроме Чуви, он не имеет доступа в машинное отделение). Пассажиры имеют доступ только к пассажирскому залу, кроме Джедаев, они имеют доступ к кабине. Люк также имеет доступ к оружию. Инженеры имеют доступ к оружию и машинному отделению.

Наиболее важно, мы можем видеть, что Хан и R2D2 находятся теперь в двух местах ACL. Им нет необходимости быть уникальными. Это делает политики более ясными для читателя: “Ах, Хан и R2D2 имеют доступ к оружию и машинному залу, потому что они инженеры”.

Добавление людей

Хан идет в Город облаков, чтобы повидать Ландо и сделать некоторый ремонт. Ландо – предыдущий хозяин Сокола тысячелетия, Хан предполагает отнести его к группе “Команда”. Ландо так же предлагает услуги своего главного инженера Хонтука, для помощи в восстановлении судна, пока оно находится в доке.



Это дерево показывает как легко можно предоставлять доступ новым людям. Если бы мы использовали первоначальную матричную схему, мы должны были бы установить разрешения для каждого места на корабле и для Ханука и для Ландо. Вместо этого, мы просто добавляем их к нужным группам, в результате чего, их доступ неявно и легко определен.

Устранение конфликтов

Что случится, если мы добавим Чуви к списку инженеров?

По умолчанию:	[DENY: ALL] (запретить: все)
Пассажиры Сокола тысячелетия	
— Команда	[ALLOW: ALL] (разрешить: все)
— Хан	
— Чуви	[DENY: Engines] (запретить: машинный зал)
— Ландо	
— Пассажиры	[ALLOW: Lounge] (разрешить: пассажирский зал)
— Джедаи	[ALLOW: Cockpit] (разрешить: кабина)
— Оби-Ван	
— Люк	[ALLOW: Guns] (разрешить: оружие)
— R2D2	
— СЗРО	
— Инженеры	[ALLOW: Engines, Guns] (разрешить: маш.зал, оружие)
— Хан	
— R2D2	
— Ханук	
— Чуви	

Это делает доступ Чуви к машинному залу неоднозначным, потому что теперь есть две дорожки от корня дерева до Чуви. Если компьютер последует по одной дорожке, то результат будет “DENY” (запрещаю), а если он последует за другой дорожкой, то результат будет “ALLOW” (разрешаю). Так все же, ему запрещен или разрешен доступ?

PhpGACL предупредит вас, если вы сгруппируете ARO таким образом, что доступ ARO к произвольному ACO будет неоднозначным. Но решать конфликт придется вам.

Если при такой неоднозначности спросить компьютер “Имеет ли Чуви доступ к машинному залу?”, то ответ будет “ALLOW” (разрешаю), потому что система вернет вам последний измененный параметр (это политика phpGACL). В этом случае результат “ALLOW”, потому что разрешение “ALLOW: Engines, Guns” (разрешить: машинный зал, оружие), назначенное на группу “Инженеры”, является более свежим, по сравнению с “DENY: Engines” (запретить: машинный зал) группы “Чуви”.

В случае, если ACL содержит неоднозначности, такую ACL называют **непоследовательной**. Непоследовательная ACL может быть очень опасна, вы можете невольно обеспечить доступ несоответствующим людям. Когда phpGACL предупреждает вас о том, что ACL непоследовательна, разрешите конфликт как можно более оперативно, чтобы сразу восстановить последовательность.

Чтобы решить конфликт, вы могли сделать следующее:

- удалить дерективу “DENY: Engines” у группы “Чуви” в группе “Команда”;
- добавить дерективу “DENY: Engines” у группы “Чуви” в группе “Инженеры”;
- удалить Чуви из группы “Инженеры”, так как Хан все равно не считает его достойным инженером.

Ха выбрал последний вариант, он удалил Чуви из группы “Инженеры”.

Обозначение объектов доступа

phpGACL уникально идентифицирует каждый объект доступа (ARO, AXO, ACO) комбинацией их двух ключевых слов и их типа объекта доступа.

Кортеж “(Тип объекта доступа, Раздел, Значение)” уникально идентифицирует любой объект доступа.

Первый элемент кортежа – тип объекта доступа (ARO, AXO, ACO).

Второй элемент кортежа, названный **Разделом**, является, заданной пользователем, строкой, которая называет общую категорию объекта доступа. Объекты с несколькими доступами могут иметь тоже самое название Раздела. Название раздела должно быть короткое, но описательное (наглядное). Это используется в пользовательском интерфейсе в списках выбора, поэтому логично не делать их слишком длинными.

Разделы сохраняются в ячейке namespace. Разделы не имеют никакого отношения к группам или ARO / AXO-деревьям – они просто механизм, для того чтобы помочь поддерживать большее количество объектов доступа.

Третий элемент кортежа – определенное пользователем, название для объекта доступа, и называется **Значение**. Значение не может содержать пробелы (однако, Раздел – может).

Названия Разделов и Значение регистрозависимы.

Примечание: обычно спрашивают, почему для идентификации объектов доступа используются строки а не целые числа, которые, якобы, кажутся быстрее. Ответ – для ясности. На много более легко понять:

```
acl_check('system', 'login', 'user', 'john_doe');
```

 чем:

```
acl_check(10, 21004, 15, 20304);
```

Так как из контекста часто очевидно к какому типу объекта доступа мы обращаемся, интерфейс для phpGACL (и эта документация) опускает тип объекта доступа и использует формат “Раздел -> Значение” для показа объекта доступа. Однако, программный интерфейс приложения требует “Раздел” объекта доступа и “Значение”, которые должны быть определены в отдельных аргументах функции (тип объекта доступа обычно неявен в описании аргумента).

Пример ACO “Раздел -> Значение”:

- “Floors -> 1st” (“Этажи -> 1-й”);
- “Floors -> 2nd” (“Этажи -> 2-й”);
- “Rooms -> Engines” (“Комнаты -> машинный_зал”).

Пример ARO “Раздел -> Значение”:

- “People -> John_Smith” (“Люди -> Джон_Смит”);
- “People -> Cathy_Jones” (“Люди -> Кэти_Джонс”);
- “Hosts -> sandbox.something.com” (“Хосты -> sandbox.something.com”).

Пример использования API:

- `acl_check(aro_section, aro_value, aco_section, aco_value);`
- `acl_check('People', 'John_Smith', 'Floor', '2nd');`

Примеры правильного именования:

- “ACO - Frob > Flerg”, “ARO – Frob -> Flerg” (Раздел и Значение являются одинаковыми, но это хорошо, поскольку namespaces разные для разных типов объектов доступа);
- “ACO – Frob -> Flerg”, “ACO – Frob -> Queeple” (тип объекта доступа и название Раздела одинаковые, это хорошо, поскольку у них разные Значения);
- “AXO – Frob Hrung -> Flerg” (Разделы могут содержать пробелы).

Пример неправильного именования:

- “ACO – Frob -> Flerg”, “ACO – Frob -> Flerg” (тип объекта доступа – Раздел -> Значение должны быть уникальными);
- “ACO – Frob -> Flerg Habit” (Значение не может содержать пробелов).

Добавление разделов

Прежде чем добавить новый объект доступа вы должны создать Раздел. Для добавления нового раздела используйте функцию `add_object_section()`.

```
add_object_section(
    string NAME,           Короткое описание того, для чего этот Раздел
                          используется (пример, 'Levels in building'
                          ("Этажи в здании")).
    string VALUE,         Название Раздела (пример, 'Floors' ("Этажи")).
    int ORDER,           Ценность раздела, влияет на то, каким по счету в
    bool HIDDEN          списке
                          в интерфейсе пользователя появится этот Раздел.
    string GROUP_TYPE);  Показывает, должен ли этот раздел быть виден в
                          пользовательском интерфейсе.
                          Тип объекта доступа (ACO, ARO, AXO).
```

Хан создал 3 раздела для AROs “Люди”, “Чужие” и “Андроиды” Давайте посмотрим, как будет выглядеть AROs с их полными названиями:

```
Пассажиры Сокола тысячелетия
├── Команда [ALLOW: ALL]
│   ├── "Люди > Хан"
│   ├── "Чужие > Чуви" [DENY: Engines]
│   └── "Люди > Ландо"
├── Пассажиры [ALLOW: Lounge]
│   ├── Джедаи [ALLOW: Cockpit]
│   └── "Люди > Оби-Ван"
```

```

├──┬ "Люди > Люк"                [ALLOW: Guns]
  ├── "Андройды > R2D2"
  └── "Андройды > C3PO"
└── Инженеры                    [ALLOW: Engines, Guns]
    ├── "Люди > Хан"
    ├── "Андройды > R2D2"
    └── "Чужие > Хантук"

```

Разделы – только способ категоризировать объекты доступа, сделать пользовательский интерфейс более пригодным к использованию, а так же сделать функцию `acl_check()` более читаемой. Они не затрагивают путь `phpGACL` для определения прав объекта доступа. Также, разделы не могут быть вложенными, то есть вы не можете создать “Мужчин” внутри “Людей”. Вы должны будете создать раздел “Люди мужского пола” или подобный.

Множество целей

Возможно, вы будете использовать `phpGACL` сразу для нескольких целей, например, ограничить пользовательский доступ к страницам, а так же удаленный доступ к страницам вашего сервера. Эти две задачи не связаны.

`PhpGACL` предлагает реализовать это тремя различными путями:

- вы можете использовать дополнительные базы данных для хранения этих данных;
- вы можете использовать ту же самую базу данных но с другими именами таблиц (всеже, эта возможность пока не поддерживается);
- вы можете хранить объекты доступа в тех же самых таблицах, и очень тщательно управлять ими, так, чтобы они не вступали в противоречия.

Чтобы выбрать первый или второй способ (когда он будет доступен) используйте массив `$gac1_options`, когда создаете новый класс `phpGACL`. Это позволит вам определить базу и префикс для таблиц:

```

$gac1_options = array(
    'db_table_prefix' => 'gac1_',
    'db_type' => 'mysql',
    'db_host' => 'host1',
    'db_user' => 'user',
    'db_password' => 'passwd',
    'db_name' => 'gac1');

$gac1_host1 = new gac1
($gac1_options);

```

Будьте очень осторожны, если вы выберете третий вариант, поскольку `phpGACL` ничего не знает об отношениях между вашими задачами, а, следовательно, не сможет контролировать различные бессмысленные директивы политики доступа.

Например, Хан может решить запретить доступ кораблям, которые вступают в контакт с его компьютером, к каким-либо комнатам на корабле. Для этого он может создать “Корабль Люка” как внешний корабль ARO в этом же самом ARO-дереве. В результате чего, станет возможным создать

APD (правило автоматической обработки): “Корабли -> Корабль Люка” и назначить ему права “ALLOW (разрешить): “Комнаты -> пассажирский зал”. Согласитесь, это полностью бессмысленно! Чтобы помочь в разрешении таких противоречий, давайте правильное (информативное) название Разделам, чтобы они четко давали понять, чем на самом деле является объект доступа и для каких задач используются. Это должно быть очевидно для любого администратора, что бессмысленно назначать разрешения кораблям для доступа в какую-либо комнату.

Объекты расширения доступа (АХО)

Объекты расширения доступа могут добавлять третье измерение к разрешениям, которые могут формироваться в phpGACL. Мы с вами видели, как phpGACL позволяет комбинировать объекты ARO и ACO (2 измерения) для создания политики доступа. Это очень хорошо для простых разрешений, например:

Люк (ARO) просит разрешение на доступ к Оружию (ACO).

Если это все, чего вы хотите, прекрасно – АХО является полностью дополнительным (то есть может отсутствовать).

Но, поскольку все ACOs являются равными, это мешает справляться с ними если их очень много. Если это так, то мы можем изменить способ, которым мы смотрим на объекты доступа, для более легкого управления ими.

АХОs идентичны AROs во многих отношениях. Есть дерево АХО (отдельное от дерева ARO), со своими собственными Группами и АХОs. Когда вы будете иметь дело с АХО, думайте, что АХО берет на себя старую роль ACO (то есть является “вещами, к которым требуется доступ”), и изменяют представление ACOs с “вещи, к которым требуется доступ” на “действие, которое требуется совершить”.

Посмотрим только на ARO и ACO:

- ARO – вещи, просящие доступ;
- ACO – вещи, к которым нужен доступ.

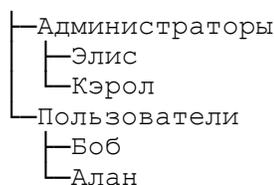
Посмотрим на ARO, ACO и АХО:

- ARO – вещи, которые просят доступ;
- ACO – действия, которые требуется совершить
- АХО – вещи, к которым нужен доступ.

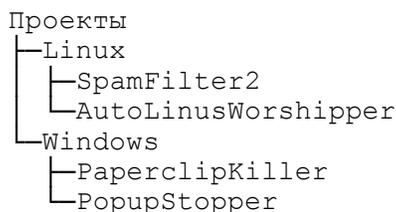
Пример:

Менеджер сайта пытается управлять доступом к проектам на вебсайте. Дерево ARO состоит из всех пользователей:

Вебсайт



Проекты организованы по принципу используемой операционной системы в Группях АХО:



Действия, которые могут выполняться над каждым объектом - “Просмотр” и “Редактирование” - это АСОs.

И так, мы хотим, чтобы Боб имел возможность просмотра всех проектов Linux. Это возможно при помощи добавления ADP (автоматической обработки) связки ARO Боба с АСО “Просмотр” и АХО Linux. Теперь мы можем задать вопрос:

“Боб” (ARO) просит действия “Просмотр” (АСО) для проекта “Linux” (АХО).

Обращаю ваше внимание на то, что АХО является дополнительным, если вы не определите АХО, вызывая `acl_check()`, то ADP (автоматическая обработка) осуществится без учета АХО. Однако, если ADP (автоматическая обработка) существует только с АХО, а вы вызовете `acl_check()` без указания АХО, вы потерпите неудачу.

И так, если вы при вызове `acl_check()` определяете АХО, то эта функция будет искать в ACL только содержание АХО. Если никакой АХО не определен, только будет рассмотрены только ACL. Это (в теории) дает нам небольшое увеличение скорости работы.

Инсталляция

Базовая настройка

1. Распакуйте архив дистрибутива в корень или нужную вам папку сайта. Возможно, вы захотите переименовать ее во что-то более подходящее.

```
karsten@tolkien:~/Work/phpgacl> tar xvfz ../mgw/libs/phpgacl-3.1.0.tar.gz
phpgacl-3.1.0/
phpgacl-3.1.0/CHANGELOG
phpgacl-3.1.0/AUTHORS
phpgacl-3.1.0/Cache_Lite/
phpgacl-3.1.0/Cache_Lite/Cache_Lite.php
phpgacl-3.1.0/Cache_Lite/Hashed_Cache_Lite.php
phpgacl-3.1.0/Cache_Lite/LICENSE
phpgacl-3.1.0/COPYING.lib
phpgacl-3.1.0/CREDITS
phpgacl-3.1.0/FAQ
phpgacl-3.1.0/MANUAL.HTML
```

2. Редактируйте файл `phpgacl/gacl.class.php`. Установите `db_type`, `db_host`, `db_user`, `db_password` и `db_name`, которые вы предполагаете использовать.

```
class gacl {
    // --- Private properties ---

    /*
     * Enable Debug output.
     */
    var $_debug = FALSE;

    /*
     * Database configuration.
     */
    var $_db_table_prefix = 'gacl_';
    var $_db_type = 'mysql'; //mysql, postgres7, sybase, oci8po
    var $_db_host = 'localhost';
    var $_db_user = 'root';
    var $_db_password = '';
    var $_db_name = 'gacl';

    /*
     * NOTE:      This cache must be manually cleaned each time ACL's are modified.
     *            Alternatively you could wait for the cache to expire.
     */
    var $_caching = FALSE;
    var $_force_cache_expire = TRUE;
    var $_cache_dir = '/tmp/phpgacl_cache'; // NO trailing slash
    var $_cache_expire_time=600; //600 == Ten Minutes
```

Теперь необходимо отредактировать `phpgacl/admin/gacl_admin.inc.php`, внося в него ту же самую информацию. Этот файл нужен для административной части скрипта.

Причина, почему необходимо дважды ввести одинаковые данные, заключается в том, что класс `gacl.class.php` намного меньше по объему, чем весь программный интерфейс. И нет необходимости тянуть за собой все скрипты, если вам всего лишь необходимо вызвать функцию `acl_check()`.

3. Создайте базу данных с именем, которое вы задали в `db_name`.
4. Запустите `http://your_site.net/phpgacl/setup.php`. Этот скрипт создаст необходимые таблицы. Не бойтесь сложной информации, скрипт покажет вам только сообщения об успешной работе.

Testing database connection...

Success! Connected to "mysql" database on "localhost".

Testing database type...

Success! Compatible database type "mysql" detected!
Making sure database "my_app_db" exists...
Success! Good, database "my_app_db" already exists!
Attempting to create tables in "my_app_db"
Attempting to create table: "acl"...
Success! Table "acl" created successfully!
Attempting to create table: "aco"...
Success! Table "aco" created successfully!
Attempting to create table: "aco_map"...
Success! Table "aco_map" created successfully!
Attempting to create table: "aco_sections"...
Success! Table "aco_sections" created successfully!
Attempting to create table: "axo_groups"...
Success! Table "axo_groups" created successfully!
Attempting to create table: "groups_axo_map"...
Success! Table "groups_axo_map" created successfully!
Attempting to create table: "axo_groups_map"...
Success! Table "axo_groups_map" created successfully!
Attempting to create table: "axo_groups_path"...
Success! Table "axo_groups_path" created successfully!
Attempting to create table: "axo_groups_path_map"...
Success! Table "axo_groups_path_map" created successfully!
Success! Installation Successful!!!

IMPORTANT

Please make sure you create the `<phpGACL root>/admin/smarty/templates_c` directory, and give it **write permissions** for the user your web server runs as. [Go here!](#) to get started.

5. Следуя замечению на картинке (Перевод замечания на картинке: Пожалуйста, создайте директорию `/admin/templates_c` и установите на нее права доступа. Нажмите "Go here!" для начала работы), создайте необходимую директорию. Права доступа должны позволять запись в эту директорию.
6. Нажмите на "Go here!" и вы перенесетесь сюда: `http://your_site.net/admin/acl_admin.php`.

Дополнительные настройки

Если вы уже используете **ADODB** в своих скриптах, то вы можете указать phpGACL использовать уже имеющуюся установку:

1. Отредактируйте `phpgacl/gacl.class.php` переменную `ADODB_DIR`, указав в ней путь к установленной библиотеке.

2. Переименуйте директорию phpgacl/adodb в какую-нибудь другую, например adodb_x и перезапустите phpgacl/admin/acl_admin.php для просмотра изменений.
3. Удалите директорию adodb, установленную вместе с phpGACL.

Если вы уже используете Smarty в своих скриптах, то вы можете указать phpGACL использовать уже имеющуюся установку:

1. Отредактируйте скрипт phpgacl/admin/gacl_admin.inc.php, установив нем следующие переменные: \$smarty_dir и \$smarty_compile_dir, чтобы они указывали на каталог, где у вас уже установлен Smarty. Переместите директорию с шаблонами phpGACL в другую директорию (например в ту, которую вы на данный момент уже используете).
2. Переименуйте папку phpgacl/smarty в другую, например smarty_x. И перезапустите phpgacl/admin/acl_admin.php для просмотра изменений.
3. Удалите папку smarty инсталляции phpGACL.

Как можно переместить phpGACL файлы из моего дерева вебсайта с сохранением связей с деревом для администрирования?

1. Перейдите в корень вашего вебсайта.
2. Переместите директорию phpGACL в необходимую вам директорию и создайте симлинк (ссылку) на административную директорию там, где вы хотите его запускать.

```
mv phpgacl/ /www/includes_directory  
ln -s /www/includes_directory/phpgacl/admin/  
gacl
```

3. Теперь попробуйте запустить phpgacl. Если он не стал работать, то, возможно, ваш сайт не поддерживает симлинки(ссылки).

Использование phpGACL в вашей программе

Базовое использование

Этот пример показывает основной способ использования phpGACL в вашей программе. Он так же показывает использование абстрактного класса для доступа к базам данных ADOdb. Пример иллюстрирует простой способ проверки логина и пароля.

```
// Подключаем базовый API
include('phpgacl/gacl.class.php');
$gacl = new gacl();
$username = $db->quote($_POST['username']);
$password = $db->quote(md5($_POST['password']));
$sql = 'SELECT name FROM users WHERE name=';
$sql .= $username.' AND password='.$password;
$row = $db->GetRow($sql);
if($gacl->acl_check('system', 'login', 'user', $row
['name'])) {
    $_SESSION['username'] = $row['name'];
    return true;
}
else
    return false;
```

Вы видите только один вызов `acl_check()`, что же он означает?

- Проверяет ARO-объект `$row['name']` в ARO-разделе 'user'.
- А также ACO-объект 'login' в ACO-разделе 'system'.

Профессиональное использование

Использование административного интерфейса

phpGACL ACL List [ARO Group Admin] [AXO Group Admin] [ACL Admin] [ACL Test] [ACL Debug]										
Filter										
	ACO	ARO	ARO Group	AXO	AXO Group	Return Value	Access	Enabled		
Section	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	Any	Any
Object	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Filter										
< << >> >										
ID	Section > ACO	Section > ARO	ARO Group	Section > AXO	AXO Group	Return Value	Access	Enabled	Updated Date	Functions
11	1. System > Email Forgotten PW 2. System > Enable - Login		1. Families				ALLOW	Yes	01-Dec-02 05:12:39	[Edit] <input type="checkbox"/>
Note:										
12	1. Projects > Add 2. Projects > View		1. Families				ALLOW	Yes	01-Dec-02 05:12:46	[Edit] <input type="checkbox"/>
Note:										
13	1. Projects > Delete 2. Projects > Edit	1. Users > Jane Doe 2. Users > John Doe					ALLOW	Yes	01-Dec-02 05:12:47	[Edit] <input type="checkbox"/>
Note:										
14	1. Projects > Add	1. Users > Jane Doe					DENY	Yes	01-Dec-02 09:12:47	[Edit] <input type="checkbox"/>
Note:										
15	1. Projects > Edit		1. Smith Family	1. Projects > Accounting (ID: 5598)			ALLOW	Yes	01-Dec-02 09:12:48	[Edit] <input type="checkbox"/>
Note:										
29	1. Projects > Add Limit	1. Users > Jane Doe				15	ALLOW	Yes	31-Jan-03 19:01:51	[Edit] <input type="checkbox"/>
Note:										
< << >> >										
<input type="button" value="Delete"/>										

phpGACL - Generic Access Control List
Copyright © 2002 [Mike Benoit](#)

Отсутствует в оригинальной документации

Описание API

ACL

add_acl()

Добавляет права доступа в лист контроля.

```
add_acl(  
    array ACO Ids,  
    array ARO_IDS,  
    array ARO_GROUP_IDS,  
    array AXO_IDS,  
    array AXO_GROUP_IDS,  
    bool ALLOW,  
    bool ENABLED  
    [, int ACL_ID])
```

Возвращает:

int ACL_ID, если все прошло хорошо и FALSE, в случае неудачи.

edit_acl()

Редактирует права доступа, которые уже установлены в листе контроля.

```
edit_acl (  
    array ACO IDs,  
    array ARO_IDS,  
    array ARO_GROUP_IDS,  
    array AXO_IDS,  
    array AXO_GROUP_IDS,  
    bool ALLOW,  
    bool ENABLED  
    [, int ACL_ID] )
```

Возвращает:

int ACL_ID если все прошло хорошо и FALSE, в случае неудачи.

del_acl()

Удаляет права доступа, которые уже заданы в листе контроля.

```
del_acl (  
    int ACL ID)
```

Возвращает:

TRUE если все прошло успешно, FALSE, в случае неудачи.

Groups (Группы)

get_group_id()

Возвращает ID группы.

```
get_group_id (  
    string GROUP_NAME)  Тип объекта доступа (ARO, ACO, AXO)
```

Возвращает:

int GROUP_ID в случае успеха, FALSE, если неудача.

get_group_parent_id()

Возвращает ID непосредственного родителя группы.

```
get_group_parent_id (  
    int GROUP_ID)
```

Возвращает:

int GROUP_PARENT_ID если удача, FALSE – если неудача.

add_group()

Добавляет группу в дерево групп.

```
add_group (  
    string NAME  
    [, int GROUP_PARENT_ID]  
    [, string OBJECT_TYPE])
```

Возвращает:

int GROUP_ID если удача, FALSE если неудача.

get_group_objects()

Возвращает вписок всех объектов отнесенных к группе.

```
get_group_aro (  
    int GROUP_ID,  
    string GROUP_TYPE)
```

Возвращает:

array SECTION_VALUE, VALUE если удача, FALSE если неудача.

add_group_object()

Соотносит ARO с группой.

```
add_group_aro (  
    int GROUP_ID,  
    string OBJECT_SECTION_VALUE,  
    string OBJECT_VALUE,  
    string GROUP_TYPE)  Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

int TRUE если удача, FALSE если неудача.

del_group_object()

Удаляет ARO из группы.

```
del_group_aro (  
    int GROUP_ID,  
    string OBJECT_SECTION_VALUE,  
    string OBJECT_VALUE,  
    string GROUP_TYPE)  Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

int TRUE если удача, FALSE если неудача.

edit_group()

Редактирует группы

```
edit_group (  
    int GROUP_ID,  
    string NAME,  
    int GROUP_PARENT_ID,  
    string GROUP_TYPE)  Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

int TRUE если удача, FALSE если неудача.

del_group()

Удаляет группу, перераспределяет детей (если есть) или удаляет их.

```
del_group (  

```

```
int GROUP_ID,  
bool REPARENT_CHILDREN,  
string GROUP_TYPE) Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

int TRUE если удача, FALSE если неудача.

Access Objects (ARO/ACO/AXO) (Объекты доступа)

Этот раздел API управляет объектами доступа, такими как ACO, ARO, AXO.

get_object()

Возвращает массив информации обо всех объектах указанного типа.

```
get_object (  
  [string SECTION_VALUE], Необязательный, название Раздела (как фильтр)  
  bool RETURN_HIDDEN, Возвращать ли объекты с типом "скрытый"  
  string GROUP_TYPE) Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

array OBJECT_ID если удача, FALSE если неудача.

get_object_data()

Возвращает массив информации об объекте доступа с конкретным ID.

```
get_object_data (  
  int OBJECT_ID,  
  string GROUP_TYPE) Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

array (section_value, value, order_value, name) если удача, FALSE если неудача.

get_object_id()

Возвращает ID объекта.

```
get_object_id (  
  string OBJECT_SECTION_VALUE,  
  string OBJECT_VALUE,  
  string GROUP_TYPE) Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

int OBJECT_ID если удача, FALSE если неудача.

get_object_section_value() Возвращает ID раздела к которому приписан объект.

```
get_object_section_value (  
    int OBJECT_ID,  
    string GROUP_TYPE)  Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

int SECTION_VALUE если удача, FALSE если неудача.

add_object()

Добавляет объект.

```
add_object (  
    string SECTION_VALUE,  
    string NAME,  
    string VALUE,  
    int ORDER,  
    bool HIDDEN,  
    string GROUP_TYPE)  Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

array OBJECT_ID если удача, FALSE если неудача.

edit_object()

Редактирование объекта.

```
edit_object (  
    string SECTION_VALUE,  
    string NAME,  
    string VALUE,  
    int ORDER,  
    bool HIDDEN,  
    string GROUP_TYPE)  Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

array OBJECT_ID если удача, FALSE если неудача.

del_object()

Удаляет объект.

```
del_object (  
    int OBJECT_ID,  
    string GROUP_TYPE)
```

```
int OBJECT_ID,  
string GROUP_TYPE,  Тип объекта доступа (ARO, AXO, ACO)  
bool ERASE)
```

Возвращает:

int TRUE если удача, FALSE если неудача.

Access Object Sections (Разделы объектов доступа)

Этот раздел API позволяет управлять Разделами, которые являются частью уникального названия объекта доступа (см. “Разделы” для дополнительной информации).

get_object_section_section_id()

Возвращает ID существующего Раздела. Вы должны указать название Раздела или его значение, или оба параметра.

Если вы определите только название, то возможны неоднозначные результаты (так как название не обязано быть уникальным). В этом случае вы получите сообщение об ошибке.

```
get_object_section_section_id (  
  string NAME,          Короткое описание того, для чего предназначен Раздел  
                       (например, Этажи в здании")  
  string VALUE,        Название Раздела (например, "Этаж")  
  string GROUP_TYPE)  Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

int SECTION_ID если удача, FALSE если неудача.

add_object_section()

Добавляет новый Раздел объектов.

```
add_object_section(  
  string NAME,          Короткое описание того, для чего предназначен Раздел  
                       (например, Этажи в здании")  
  string VALUE,        Название Раздела (например, "Этаж")  
  int ORDER,           Ценность Раздела. Учитывается в пользовательском  
интерфейсе при выводе списка выбора.  
  bool HIDDEN,        TRUE - Раздел будет скрыт из списка в пользовательском  
интерфейсе.  
  string GROUP_TYPE)  Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

int SECTION_ID если удачно, FALSE если неудачно.

edit_object_section()

Изменяет атрибуты Раздела. Функция не может изменить тип объекта доступа (для дополнительной информации смотрите add_object_section).

```
edit_object_section (
    int OBJECT_SECTION_ID,   ID раздела (вы можете получить его при помощи
                             функции get_object_section_section_id).
    string NAME,             Новое название Раздела.
    string VALUE,            Новое значение Раздела.
    int ORDER,              Новая ценность раздела.
    bool HIDDEN,            Новый статус раздела (скрыт или нет)
    string GROUP_TYPE)      Тип объекта доступа (ARO, AXO, ACO)
```

Возвращает:

TRUE если удача, FALSE если неудача.

del_object_section()

Удаляет Раздел. Все объекты Раздела также будут удалены!!!

```
del_object_section (
    int SECTION_ID,         ID Раздела
    string GROUP_TYPE,      Тип объекта доступа (ARO, AXO, ACO)
    bool ERASE)             Если TRUE - все объекты Раздела будут удалены.
                             Если FALSE, то вы получите сообщение об ошибке в случае,
                             если Раздел содержит объекты, Раздел удален не будет,
                             если Раздел не содержит объектов, он будет удален.
```

Возвращает:

TRUE если удача, FALSE если неудача.
