

Zend_Acl - Simon Mundy

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Acl Component Proposal

Proposed Component Name	Zend_Acl
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Acl
Proposers	Simon Mundy
Revision	0.2 - Finalised proposal (wiki revision: 11)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
- ACL - Football match
- ACOs - The ground
- AROs - The groups
- Basic setup
9. Class Skeletons

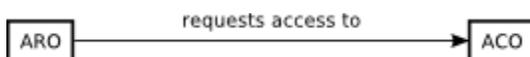
1. Overview

Zend_Acl provides lightweight and flexible access control list (ACL) functionality. In general, an application may utilize such functionality to control access to certain protected objects by other requesting objects.

In this proposal:

- an **Access Control Object (ACO)** is an object to which access is controlled.
- an **Access Request Object (ARO)** is an object that may request access to an ACO.

For example, if a person requests access to a car, then the person would be the requesting ARO, and the car would be the ACO, since its access is under control.



Through the specification and use of an access control list (ACL), an application may control how requesting objects (AROs) are granted access to protected objects (ACOs).

2. References

- [phpGACL](#)
- [ZF Developer Wiki](#)

3. Component Requirements, Constraints, and Acceptance Criteria

- No database backend required - complete PHP implementation
- Permits explicit allow-all, allow-specific, neutral, deny-specific, deny-all for all groups
- Authorization queries must be computed with efficiency and speed
- Performance penalties, where reasonably unavoidable, should be isolated to ACL data structure modifications
- Instances must be serializable (e.g., for caching ACL data structures)

4. Dependencies on Other Framework Components

- `Zend_Exception`

5. Theory of Operation

The goal of `Zend_Acl` was to create a simple, flexible means of creating ACLs, free from the dependencies of Databases, template engines or complex administration tools. The starting point was `phpGACL`, as its 'allow,deny' and inheritance scheme is straightforward and allows huge flexibility for implementation.

`Zend_Acl` builds on the tree-based model of `phpGACL`, providing a means to add multiple ACOs that may represent 'areas' or 'domains' of control within the ACL. Objects under access control may be organized from general (at the tree root) to specific (toward tree leaves). This approach allows for creating very fine-grained access controls, where desirable, with minimal effort.

Each ACO may also be assigned its own set of 'actions' or 'permissions' (e.g., add, edit, delete) that can be used to determine the level of access each requesting ARO would have. Actions may be assigned to ACOs simply by calling `allow()` or `deny()` on the ACO in question.

AROs may be organized in different ways, according to the particular needs of an application. To accommodate these various organizations, the ARO registry of `Zend_Acl` is represented by a [directed acyclic graph \(DAG\)](#) data structure. This approach supports having multiple roles or groups, and AROs may also inherit permissions from multiple AROs.

To illustrate by example, an application may need access roles for administrators, members, and the general public. Each of these roles would have a corresponding ARO in the access control list. The general public would have the least access privileges, members would have additional permissions (e.g., to edit their profiles), and administrators would have exclusive permissions to perform administrative operations (e.g., blocking abuse). Suppose we also have an ARO for each login user identity. Each of these "user" AROs may inherit permissions that have been assigned to one or more of the "role" AROs previously mentioned (e.g., administrators). In this way we can add users to, and remove users from, such roles without having to otherwise modify the access controls.

6. Milestones / Tasks

zone: Missing {zone-data:milestones}

7. Class Index

- `Zend_Acl`
- `Zend_Acl_Aro`
- `Zend_Acl_Aro_Registry`
- `Zend_Acl_Permission`

8. Use Cases

At its most simplistic level, Zend_Acl can provide the bare necessities of an authorisation scheme. To illustrate, we'll assume the following 'real-world' scenario:-

ACL - Football match

ACOs - The ground

- Stadium
- Public seating
- Reserved seating
- Corporate seating
- Coaches box
- Dressing rooms
- Pitch

AROs - The groups

- Public (or '_default' group) - unreserved seats or reserved seats
- Guests - members of the public who are invited to a corporate box
- Coaching staff
- Players
- Club officials

Basic setup

We create an ACL instance and add our AROs, adding inheritance where necessary.

```
$acl = new Zend_Acl();

$aros = $acl->aroRegistry();
$aros->add('public');
$aros->add('reserved', $aros->public);
$aros->add('guest', $aros->reserved);
$aros->add('staff');
$aros->add('official', $aros->staff);
$aros->add('coach', $aros->staff);
$aros->add('player', $aros->staff);
```

Now we start adding the expected rules typical to a game of football

```
$acl->deny(); // Automatic whitelist
$acl->seating->allow('public');
$acl->seating->north->allow('reserved'); // Extra $$$ buys you good seats
$acl->seating->south->allow('guest'); // Know someone and get better seats!
$acl->pitch->allow(array('coach', 'player'));
$acl->dressingrooms->allow(array('staff', 'guest'));
$acl->coachesbox->allow('staff');
$acl->coachesbox->deny('official'); // The coach and the president don't get on well
```

The above example demonstrates how a moderately complex ACL can be created with a minimum of effort. However, if you wanted to refine these generic rules so that the AROs can perform only certain actions, then you can start getting selective

```
$acl->coachesbox->deny(null, 'talk');
$acl->coachesbox->allow('coach', 'talk');
$acl->dressingrooms->allow('guest', array('autograph', 'talk'));
$acl->seating->deny(null, 'stand');
```

Now it's time to use your ACL during the execution of your script. A simple 'valid' method provides a boolean answer to each of your queries

```

if ($acl->seating->valid('public')) { print "Public can sit here"; }
if (!$acl->seating->south->valid('public')) { print "Please return to the east side of the ground,
sir"; }
if ($acl->pitch->goals->valid('player')) { print "...and he goes towards goals!"; }
if (!$acl->coachesbox->valid('staff', 'talk')) { print "Keep it down! Coach is thinking!"; }

```

Zend_Acl will assume that non-specific AROs or contexts will apply to **any** member, so you can provide null values to these arguments as you add permissions. However, if you add specific 'allow' or 'deny' actions to an ACO, you are effectively providing only limited access to that ACO. You can, however, add multiple permissions to the same ACO to different AROs to ensure your permissions are as fine-grained as you require. Zend_Acl will also check for conflicts when adding permissions to remove the likelihood of 'deadlocks'.

9. Class Skeletons

```

<?php

class Zend_Acl
{
    /**
     * Default group id
     */
    const ARO_DEFAULT = '_default';

    /**
     * Default path
     */
    const PATH_DEFAULT = '_default';

    /**
     * Default delimiter for paths
     */
    const PATH_DELIMITER = '/';

    /**
     * Magic catch-all keyword
     */
    const ACO_CATCHALL = '__ALL__';

    /**
     * Default permission in case of final neutral result
     */
    const PERM_DEFAULT = false;

    /**
     * Modes for adding permissions to a permission container
     */
    const MODE_SET = 1;
    const MODE_ADD = 2;
    const MODE_REMOVE = 3;
    const MODE_UNSET = 4;

    /**
     * Path name for ACO
     * @var string
     */
    protected $_path;

    /**
     * Permissions for this ACO.
     * @var Zend_Acl
     */
    protected $_perm;

    /**

```

```

    * Parent ACO.
    * @var Zend_Acl
    */
protected $_parent;

/**
 * All children of this ACO
 * @var array
 */
protected $_data = array();

/**
 * Class constructor
 *
 * A reference to the parent object is created if supplied.
 *
 * @param Zend_Acl $parent
 */
public function __construct($parent = null, $path = '_default')

/**
 * Retrieve reference to ACO via 'path' property
 *
 * If the path exists then return a reference to it, otherwise return a
 * reference to self. This means that permissions can be implied for a path
 * rather than explicitly set (they infer an inherited permission).
 *
 * @param string $path
 * @return Zend_Acl
 */
public function __get($path)

/**
 * Create path
 *
 * Add a new path to this ACL.
 *
 * @param string $name
 * @return void
 */
public function __set($path, Zend_Acl $value)

/**
 * Retrieve the global ARO registry
 *
 * @return Zend_Acl_Aro
 */
public function aroRegistry()

/**
 * Test permissions for a path
 *
 * Retrieve permissions for an ACO based on the ARO, current context
 * and an optional path. The optional path parameter allows for a top-down
 * search from a root - if not supplied, then this instance is used as a
 * target. A 'null' context will return true if no explicit permissions
 * are set to 'deny' for the specified group on the target ACO.
 *
 * @param string $id
 * @param string $context
 * @param string $path
 * @throws Zend_Acl_Exception
 * @return boolean
 */
public function valid($aro = Zend_Acl::ARO_DEFAULT, $context = null, $path = null)

/**
 * Assert validity of container
 *
 * Convenience method to providing exception handling for ACLs

```

```

* Parameters are identical to valid()
*
* @throws Zend_Acl_Exception
* @return true
*/
public function assertValid()

/**
 * Returns the ACL's parent object
 *
 * @return Zend_Acl|null
 */
public function getParent()

/**
 * Returns the ACL's child nodes
 *
 * @return Zend_Acl|null
 */
public function getChildren()

/**
 * Returns the ACL's path
 *
 * @return string
 */
public function getPath()

/**
 * Returns an array of AROs that can access the ACL
 *
 * This function will determine which AROs - from either a list of AROs or
 * the entire ARO registry - have access to the current ARO.
 *
 * AROs can be supplied either an an ARO object, an array of ARO objects,
 * a string id or an array of string ids. If the ARO parameter is left empty
 * then the ARO registry is used to return all members.
 *
 * To allow fine-grain control, a specific context can also be used to
 * validate the AROs
 *
 * An array of ARO objects is returned upon success or empty
 *
 * @param mixed $aro
 * @param string $context
 * @return array
 */
public function getValidAro($context = null, $aro = null)

/**
 * Sets allow permissions to the ACL
 *
 * Each parameter can be a string or a numeric array of values to allow
 * assignment to many parameters at once. The $path can use a forward slash
 * delimiter to indicate a nested relative path
 *
 * @param mixed $aro
 * @param mixed $value
 * @param mixed $path
 * @return Zend_Acl Provides a fluent interface
 */
public function allow($aro = Zend_Acl::ARO_DEFAULT, $value = null, $path = null)

/**
 * Sets deny permissions to the ACL
 *
 * @param mixed $aro
 * @param mixed $value
 * @param mixed $path
 * @return Zend_Acl Provides a fluent interface

```

```

*/
public function deny($aro = Zend_Acl::ARO_DEFAULT, $value = null, $path = null)

/**
 * Removes allow permissions from the ACL
 *
 * Removes explicit allow permissions from the ACL whilst retaining existing
 * values. If no $value is passed, all explicit permissions are removed.
 *
 * @param mixed $aro
 * @param mixed $value
 * @param mixed $path
 * @return Zend_Acl Provides a fluent interface
 */
public function removeAllow($aro = Zend_Acl::ARO_DEFAULT,
                           $value = Zend_Acl::ACO_CATCHALL,
                           $path = Zend_Acl::PATH_DEFAULT)

/**
 * Removes deny permissions from the ACL
 *
 * @param mixed $aro
 * @param mixed $value
 * @param mixed $path
 * @return Zend_Acl Provides a fluent interface
 */
public function removeDeny($aro = Zend_Acl::ARO_DEFAULT,
                           $value = Zend_Acl::ACO_CATCHALL,
                           $path = Zend_Acl::PATH_DEFAULT)

/**
 * Returns allow permissions for the current ACL
 *
 * @return Zend_Acl_Permission Provides a fluent interface
 */
public function getAllow()

/**
 * Returns deny permissions for the current ACL
 *
 * @param mixed $path
 * @return Zend_Acl_Permission Provides a fluent interface
 */
public function getDeny()

/**
 * Removes an ACL node
 *
 * If a path is provided and exists, it will be destroyed. If no path is
 * provided then the current ACL will instead be removed from its parent
 * (if the current ACL is not root)
 *
 * @param string $path
 * @return boolean
 * @throws Zend_Acl_Exception
 */
public function remove($path = null)

/**
 * Removes an ARO from current node and all children
 *
 * @param mixed $aro
 * @param mixed $context
 * @param mixed $path
 * @return boolean
 */
public function removeAro($aro, $context = Zend_Acl::ACO_CATCHALL, $path = null)

/**
 * Removes an ARO from current node and all children

```

```

*
* @param Zend_Acl $root
* @param mixed $aro
* @param mixed $context
* @return boolean
*/
protected function _valid($root, $aro, $context)

/**
 * Adds permissions to one or more permission containers
 *
 * This method is responsible for passing contexts and groups to each
 * individual permissions container for processing. The $mode determines
 * if those contexts are to be set, added or removed
 *
 * @param mixed $acl
 * @param mixed $value
 * @param mixed $aro
 * @param mixed $path
 * @param mixed $mode
 * @return Zend_Acl Provides a fluent interface
 */
protected function _setPermission($acl, $value, $aro, $path,
                                $mode = Zend_Acl::MODE_SET)

/**
 * Retrieves permissions for a container
 *
 * @return Zend_Acl_Permission
 */
protected function _getPermission()

/**
 * Retrieves an array of ARO objects for the selected id
 *
 * @return array
 */
protected function _parseAro($id)

/**
 * Creates child paths from current ACO to destination ACO
 *
 * $path uses a forward slash to denote a nested path (@see setAllow()).
 * If the target path does not exist, a new empty ACL is created.
 *
 * @param string $path
 * @return Zend_Acl_Permission
 */
protected function _createPath($path)

/**
 * Creates new 'virtual' ACLs to target (nonexistent) path
 *
 * @param mixed $path
 * @return Zend_Acl
 */
protected function _addPath($path)

/**
 * Expand a path to retrieve target node
 *
 * @param Zend_Acl $root
 * @param mixed $path
 * @return Zend_Acl
 */
protected function _findPath(Zend_Acl $root, $path)

/**
 * Determines if the current ACL is root
 *

```

```

        * @return boolean
        */
        protected function _isRoot()
    }

class Zend_Acl_Aro
{
    /**
     * Unique id of ARO
     * @var string
     */
    protected $_id;

    /**
     * Collection of parents
     * @var array
     */
    protected $_parent = array();

    /**
     * Class constructor
     *
     * If $inherit contains a string or array of values, then iterate through
     * the parents, retrieving their ids and store in LIFO order. The first
     * element of the $_parent array will always refer to the current object
     *
     * @param string $id
     * @param mixed $inherit
     * @return void
     */
    public function __construct($id, $inherit = null)

    /**
     * Retrieves ARO id
     *
     * @return string
     */
    public function getId()

    /**
     * Retrieves ARO parent ids
     *
     * @return array
     */
    public function getParent()

    /**
     * Performs a validation on an ACO using the current ARO
     *
     * @return boolean
     */
    public function canAccess(Zend_Acl $aco, $context = null, $path = null)

    /**
     * Returns an ACO tree that the current ARO has access to
     *
     * This function will determine which AROs - from either a list of AROs or
     * the entire ARO registry - have access to the current ARO.
     *
     * AROs can be supplied either an an ARO object, an array of ARO objects,
     * a string id or an array of string ids. If the ARO parameter is left empty
     * then the ARO registry is used to return all members.
     *
     * To allow fine-grain control, a specific context can also be used to
     * validate the AROs
     *
     * An array of ARO objects is returned upon success or empty
     *
     * @param mixed $aro
     * @param string $context

```

```

    * @return array
    */
    public function getValidAco(Zend_Acl $aco, $context = null)

    protected function _getValidAco(Zend_Acl $aco, $context)

    /**
     * Add parent to current ARO
     *
     * @return Zend_Acl_Aro
     */
    protected function _addParent($parent)
}

class Zend_Acl_Permission
{
    /**
     * Contains allow and deny context values
     * @var array
     */
    protected $_context = array('allow' => array(),
                                'deny' => array());

    /**
     * Returns a score for the selected permission
     *
     * The score is factored according to an exact match for an ARO (3), a
     * match for an inherited ARO (2) or a match for an any/all ARO (1).
     *
     * @param string $type
     * @param array $aro
     * @param string $context
     * @return integer
     */
    public function score($type, Zend_Acl_Aro $aro, $context = null)

    /**
     * Returns the contents of the selected permission
     *
     * @param string $type
     * @return array
     */
    public function getPermissions($type)

    /**
     * Sets contexts for a permission
     *
     * $type represents either an 'allow' or 'deny'
     * $values represents the contexts allowed for the permission type and can
     * be supplied as a string or an array of values
     * $aro can be either a string id or an array of values to represent
     * multiple aros (and their inherited permissions)
     * $mode is provided as either set, add or remove
     *
     * If $values contains the magic value Zend_Acl::ACO_CATCHALL then all
     * nominated aros will provide an explicit match for the permission type.
     * Otherwise, as each context is set, the opposite is checked for to ensure
     * no deadlocks for permissions
     *
     * E.g. If 'admin' is provided for 'allow', then 'admin' will be removed
     * from 'deny' if it exists for the selected aro(s)
     *
     * @param string $type
     * @param mixed $value
     * @param array $aro
     * @param integer $mode
     * @return integer
     */
    public function setValues($type, $value, $aro, $mode = Zend_Acl::MODE_ADD)

```

```

/**
 * Parses context value
 *
 * Ensures that the magic Zend_Acl::ACO_CATCHALL value is returned as a
 * single array (as it overrides all other explicit contexts) if exists.
 * Otherwise cast the value(s) as an array for storage.
 *
 * @return string
 */
protected function _getContext($value)

/**
 * Returns a score factor for the selected Aro
 *
 * Ensures that a specific permission context is assigned a higher score
 * than an inherited permission
 *
 * @return integer
 */
protected function _getFactor(Zend_Acl_Aro $aro, $id)

/**
 * Returns the inverse to the permission type
 *
 * @return string
 */
protected function _getReverse($type)
}

class Zend_Acl_Aro_Registry
{
/**
 * ARO registry
 * @var array
 */
protected $_aro;

/**
 * Parent Aco
 * @var Zend_Acl
 */
protected $_aco;

/**
 * Registry instance
 * @var array
 */
static protected $_instance;

/**
 * Instance of Zend_Acl_Aro_Registry
 * @param Zend_Acl_Aro $group
 */
static public function getInstance()

/**
 * Class constructor
 *
 * @return void
 */
public function __construct()

/**
 * Public access to ARO registry
 *
 * @return void
 */
public function __get($aro)

/**

```

```
* Add unique ARO to registry
*
* @param Zend_Acl_Aro $aro
* @return Zend_Acl_Aro_Registry for fluent interface
*/
public function add($aro, $inherit = null)

/**
 * Remove ARO from registry
 *
 * @param Zend_Acl_Aro $aro
 * @return boolean
 */
public function remove($aro, $aco)

/**
 * Find group in registry
 *
 * If the named group does not exist, the default ARO is returned
 *
 * @param string $aro
 * @return Zend_Acl_Aro
 */
public function find($aro)

/**
 * Return registry as an array of ARO objects
 *
 * @return array
 */
public function toArray()
}
```

```
class Zend_Acl_Exception extends Zend_Exception
{
```

```
]]</ac:plain-text-body></ac:macro>
]]</ac:plain-text-body></ac:macro>
```