This plugin makes it easy to assign and track ownership and any access controls for any Moodle entity.

For an example of code using the library, check out the portfolio and repository plugins, which are found inside http://moodle.org/download/modules/portfolio.zip.

A demo using all these plugins can be freely accessed at http://portfolio.spdc.org/.

For a flowchart of the main logic of how this plugin checks a given user's access to a given entity, see access_flowchart.dia (access_flowchart.png) in this directory.

### access_control: Database Tables

<table>
<thead>
<tr>
<th>access_control</th>
<th>access_owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>+id: int</td>
<td>+id: int</td>
</tr>
<tr>
<td>+fkid: int = (opt - PK)</td>
<td>+fkid: int = (opt - PK)</td>
</tr>
<tr>
<td>+description: string = (req - PK)</td>
<td>+description: string = (req - PK)</td>
</tr>
<tr>
<td>+designee: int = (opt - PK)</td>
<td>+designee: int = (req)</td>
</tr>
<tr>
<td>+designee_type: int = (opt - PK)</td>
<td>+designee_type: int = (req)</td>
</tr>
<tr>
<td>+everybody: int = (opt)</td>
<td>+designee_description: string = (opt)</td>
</tr>
<tr>
<td>+access: int = (req - PK)</td>
<td>+timecreated: int</td>
</tr>
<tr>
<td>+yesno: bool</td>
<td>+timemodified: int</td>
</tr>
<tr>
<td>+timecreated: int</td>
<td></td>
</tr>
<tr>
<td>+timemodified: int</td>
<td></td>
</tr>
</tbody>
</table>

Optional (potentially NULL) and required fields are indicated by (opt) and (req). Primary Keys are indicated by “PK”. The following are the field definitions:
At the moment, there is a plugin architecture for API users to add arbitrary access types. This should perhaps change to use an additional database table in which types are given numeric identifiers. This would be more dynamic.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>ID of the entity to which access is being controlled. This will usually be a foreign key, but it could be any identifier value.</td>
</tr>
<tr>
<td>fkid</td>
<td>Arbitrary string specifying what type of entity is referred to by fkid. (examples: 'course', 'assignment', 'portfolio_item', 'repository_file')</td>
</tr>
<tr>
<td>description</td>
<td>ID of the entity for which/whom access is being granted or denied.</td>
</tr>
<tr>
<td>designee_type</td>
<td>Type of entity referred to be designee. (examples: ACCESS_BYUSERID, ACCESS_BYGROUPID, ACCESS_BYCOURSEID, ACCESS_BYPARENT)</td>
</tr>
<tr>
<td>everybody</td>
<td>Whether this access control applies to the designee (0), all Moodle users (1), or the whole world (2).</td>
</tr>
<tr>
<td>access</td>
<td>Numeric identifier of an access-type. Built-in defaults include all(1), read(2), write(3), delete(4), chmod(5), chown(6).</td>
</tr>
<tr>
<td>yesno</td>
<td>Whether the access is being granted(1) or denied(0).</td>
</tr>
<tr>
<td>timemodified</td>
<td>ID of the entity that is owned. This will usually be a foreign key, but it could be any identifier value.</td>
</tr>
<tr>
<td>description</td>
<td>Arbitrary string specifying what type of entity is referred to by fkid. (examples: 'course', 'assignment', 'portfolio_item', 'repository_file')</td>
</tr>
<tr>
<td>designee</td>
<td>ID of the entity referred to be designee. (examples: ACCESS_BYUSERID, ACCESS_BYGROUPID, ACCESS_BYCOURSEID, ACCESS_BYPARENT)</td>
</tr>
<tr>
<td>designee_type</td>
<td>Foreign-key reference to the description field. This enables arbitrary, abstract entities to own other entities. This provides for hierarchical ownership and hierarchical capability specification.</td>
</tr>
</tbody>
</table>

- At the moment, there is a plugin architecture for API users to add arbitrary access types. This should perhaps change to use an additional database table in which types are given numeric identifiers. This would be more dynamic.
**access_control: APIs**

For your code to use the access_control plugin, it must use the access_control APIs to create, manage, and delete entity ownership and access controls.

Almost all of the class methods have corresponding library functions that can be called with an associative array defining an object instance.

APIs (methods are bold, functions are italicized):

1. **can_designee_access()**
   
   ```
   access_can_designee_access(array)
   ```
   
   - Does the specified designee have the specified access to the specified entity?

2. **can_i_access()**
   
   ```
   access_can_i_access(array)
   ```
   
   - Does the currently authenticated user have the specified access to the specified entity?
   
3. **does_designee_own()**
   
   ```
   access_does_designee_own(array)
   ```
   
   - Does the specified designee own the specified entity?
   
4. **do_i_own()**
   
   ```
   access_do_i_own(array)
   ```
   
   - Does the currently authenticated user own the specified entity?

5. **set_owner()**
   
   ```
   set_owner()
   ```
   
   - Set the owner of the specified entity.

6. **get_owner()**
   
   ```
   access_get_owner(array)
   ```
   
   - Return the owner record of the specified entity.

7. **get_controls()**
   
   ```
   access_get_controls()
   ```
   
   - Get all the access controls that apply to the specified entity.

8. **does_designee_owner()**
   
   ```
   access_does_designee_owner(array)
   ```
   
   - Does the specified designee own the specified entity?
   
9. **do_i_owner()**
   
   ```
   access_do_i_owner(array)
   ```
   
   - Does the currently authenticated user own the specified entity?

10. **list_designeeids()**
    
    ```
    access_list_designeeids(array)
    ```
    
    - List the designees of the specified type who/which have the specified access to the specified entity.

11. **get_users()**
    
    ```
    access_get_users()
    ```
- Calls $this->list_designeeids() and fetches and returns a list of user records.

12. **get_groups()**
   - Calls $this->list_designeeids() and fetches and returns a list of group records.

13. **get_courses()**
   - Calls $this->list_designeeids() and fetches and returns a list of course records.

14. **access_change(bool)**
   - Replace or delete access_control records to change the specified designee's specified access to the specified entity.
   - The boolean parameter indicates whether you are specifying (inserting, replacing) an access control or unspecifying (deleting) an access control.

15. **delete()**
   - Safely delete all the access_control and access_owner data for a specified entity.

16. **access_control_page()**
   - Present the user with a web form to grant/deny users, groups, or courses a specified type of access to a specified entity.

17. **access_owner_page()**
   - Present the user with a web form to change the owner of a specified entity. (It is only possible to specify users, groups, and courses as owners through this UI. Specifying parent entities as owners is only done through the API.)

18. **find_user_shares(userid, byuser, bygroup, bycourse)**
   - Find all the entities of the specified description which the user does not own, but to which the user has the specified access personally, or by virtue of group membership or course enrollment/teachership.

19. **set_access_like(model(object), safe, setowner, setaccess)**
   - Set the access of the specified entity to be a copy of the access controls for the model entity.
   - If setowner is true, copy the model entity's ownership to the specified entity.
   - If setaccess is true, copy the model entity's access to the specified entity.
Examples

A) In portfolio plugin lib.php, the following code fetches some portfolio_item records and then checks them one-by-one to be sure the current user can view (read) them. (I haven't yet implemented group-checking to cut down on DB access, but this optimization is on the way.)

```php
if ($prelim_records = get_records_sql($sql)) {
    $ac_defn = array(
        'description' => PORTFOLIO_ACCESS_BYITEM,
        'access' => 'read',
    );
    if (!$ac = new access_control($ac_defn)) {
        return false;
    }
    foreach ($prelim_records as $prelim_record) {
        $ac->fkid = $prelim_record->id;
        if ($ac->can_i_access(true)) {
            $records[] = $prelim_record;
        }
    }
    return $records;
}
```

B) The repository plugin itself has a Moodle Native plugin, and that plugin implements a front-end API called set_owner:

```php
set_owner($resourceid=0, $safe=false, $ownerid=0, $owner_type=0, $owner_description='')
```

That API contains the following code, which calls through to the access_control set_owner() method:

```php
$owner = new Object;
$owner->fkid = $resourceid;
$owner->description = REPOSITORY_NATIVE_ACCESS_DESC;
$owner->designee = $ownerid;
$owner->designee_type = $owner_type;
$owner->designee_description = $owner_description;
$owner->timecreated = time();
$owner->timemodified = time();
if (!$ac = new access_control($owner)) {
    return false;
}
if (!$ac->set_owner($safe)) {
    return false;
}
return true;
```