# Conditional activity availability: proposed enhancements

*sam marshall ● 10 February 2014, v5 (community version)*

We propose to upgrade the Moodle conditional activity availability feature as part of Moodle 2.7.

This change will be developed by me; development will be funded by my employer, The Open University. **As of writing, the development time has not yet been completely agreed** – I am going ahead with investigation and prototyping, but it is possible that we might have to call a halt to development, or delay it, depending on local conditions.

## Current system

The present availability feature controls access to Moodle activities based on certain factors:

* Date (available from / until).
* Completion status of other activities.
* Grades for other activities.
* User profile fields.
* Grouping. *(Only if experimental option* enablegroupmembersonly *is turned on.)*

These are configured by fields in the activity settings form, mostly (except for grouping) under the **Restrict access** heading. You can use multiple restrictions (and in some cases, multiple instances of the same type of restriction, such as requiring a certain grade in activity A and also in activity B). If multiple restrictions are specified, all must be met in order to view the activity (AND).

An additional setting controls what happens when the activity is not available. There are two choices:

* Activity does not appear to students at all.
* Activity appears but is greyed out / not a link, and includes information about when/how it will become available.

## Main front-end changes

The main proposed change (user requirement) is this:

* Support OR and NOT conditions. At present conditions can only be AND (including all conditions selected). **MDL-34943 MDL-26873**

In order to be useful, this change implies additional requirements:

* Support condition trees, so you can do logic such as ‘user must have (completed this activity OR that other activity) AND date must be after X’.  
  *Without this, OR and NOT conditions are not very useful*.
* Improve the user interface for specifying conditions.  
  *The interface in the activity settings form is already too large and complex and it is not possible to extend it to support OR and trees without a major change.*

## Back-end changes

I propose making back-end changes (which will not be immediately visible to users) which are either required for these changes, or else would be sensible to develop at the same time:

* Change the way conditions are stored in database. At present certain conditions are stored in custom tables and others are in fields in the course-modules table. I propose changing this to a single field in the course-modules table (probably included also in the modinfo cache) which stores a JSON object containing all the required data.   
  *This is necessary in order to support the OR / condition tree feature.*
* Support pluggable conditions. At the moment, conditions are hardcoded within availabilitylib.php. Initially this choice was made for performance, but I believe we should be able to implement pluggable conditions – there are frequently requests to add other types of availability condition, and using monolithic code makes this impractical. **MDL-32927**.  
  *This is not strictly necessary right now but we will need to substantially rewrite most of this part of code so it is a good point to make this important improvement, and it will enable future enhancements which are likely to be required later. E.g.* **MDL-42991***,* **MDL-42109***.*
* Relatively trivial, but we should change the way the ‘show information greyed out’ option applies. At the moment the behaviour is inconsistent and confuses people because even if you choose to show information, the activity is still completely hidden if restricted (a) by grouping, or (b) by ‘available until’ date. We should add options so the existing functionality can be maintained, but change the default to be less surprising.  
  *There’s an MDL number for this request but I forget it.*
* Make it add a warning or prevent deletion if you delete an activity that a condition relies on. At the moment, if you make activity X conditional on activity Y being complete, then delete activity Y, there isn’t any additional warning.  
  *Just poor behaviour that should be improved while working on this area if possible.* **MDL-43868** (partial).

## Grouping behaviour changes

In addition, I propose a change to the way the grouping field currently works. I suggest that:

* The experimental admin setting enablegroupmembersonly, and the corresponding groupmembersonly option on individual activities, should be removed.
  + If any users have this experimental setting on, but do not have enableavailability (‘Enable conditional access’) , we will need to automatically turn on enableavailability during upgrade so that we can preserve their system’s behaviour.
* When groupmembersonly is turned on for an activity, we represent this instead with a grouping restriction (the grouping restriction will support ‘all groups’ pseudo-grouping, so we can still use it to implement groupmembersonly even where a grouping wasn’t selected). This can be applied automatically during upgrade.
* For activities which do not support groups, currently the grouping option still appears together with groupmembersonly; this will no longer appear.

The effect of these changes is that grouping (when used to select a set of groups, under the **Groups** heading) is decoupled from grouping (when used to restrict access, under the **Restrict access** heading). This should make the interface easier to understand and, as a consequence, you will also be able to use the Boolean logic with groupings (e.g. must belong to X grouping OR Y grouping).

## Interface detail

Diagrams below show how the feature might work. These are subject to change and do not represent the actual appearance.

### Key points

* The interface will remain in its current location (**Restrict access** heading on activity settings form) so there is no problem with locating the settings.
* When there are no restrictions (default situation) the interface will appear almost empty (representing the actual situation) – 2 rows with 1 button, instead of 9 complex rows at present.
* From Moodle 2.7, it is not necessary to provide a non-JavaScript version of the user interface. We can use more advanced features without having to implement it twice.

### Initial appearance

This is how the form will appear when you have not added any restrictions.



* When you click the **Add restriction** button, a popup menu or dialog of some kind appears that lets you select which type of restriction you wish to add.  
   I haven’t drawn it, but ideally this should be a one-click menu/dialog, to minimise the number of additional clicks over current behaviour.  
  At minimum, the options will include all the restrictions in current Moodle:
  + Date from
  + Date until
  + Grade
  + Activity completion (of another activity)
  + Grouping
  + User field

### After adding a restriction and completing it

When you add a restriction, it appears with form controls that you can use to set its values. These should be basically the same as the controls used for the available options in the current form.

#### 

In this screen, we’ve added a **Grade** restriction, then selected an activity (**Some activity**) and typed in a required grade value.

* For **Grade** restrictions, the dropdown options will be (at minimum) ‘at least’ and ‘less than’. This corresponds to the current interface with two boxes. For clarity we should probably also display the >= and < symbols along with the text of the option.
  + **Note**: This specification does not intend to detail the specific interface for each type of restriction – they are included in these diagrams only as examples. The implementation will make it possible to define all the same restrictions that you can define with the present interface.
* The X button can be used to delete the restriction.  
  This applies immediately with no confirmation. If you don’t really want it to happen, you can always cancel the form edit – none of these changes will apply unless you save the form.
* The show/hide eye icon (consistent with the standard eye icon for hiding activities entirely) controls whether the activity is visible to users who fail this condition.
  + This is equivalent to the option that used to be presented as a dropdown **While access is prevented**. There is more information about this with the following screens.
  + In this example, because the eye is open, even users who have a grade less than 70% (or no grade) will still see the activity. It will be greyed out, with information that they need to score a higher grade in **Some activity**. (Same as the existing **Show activity greyed out, with restriction information** dropdown option.)
  + If you toggle the eye icon closed, then the activity will not appear at all to users with a grade less than 70%. (Same as the existing **Hide activity entirely in the course and gradebook** option.)
* The **must** dropdown lets you switch between positive and negative conditions (see below).

### Adding a second restriction

When you add more than one restriction, the ‘AND/OR’ choice becomes visible to the user.



This screen shows what happens after the previous screen if we add a user field. The user field has not yet been completed.

* **Student must match all of the following** indicates an AND choice. To make this completely clear, we insert the word **and** between each restriction.
* The **all** dropdown has another choice **any** (or possibly **at least one of** or similar – wording to be determined during implementation). By changing the dropdown, you can instantly toggle the condition to an OR choice. (All instances of **and** between the restrictions will change.)
* The user field isn’t completed; it has a dropdown left on **Choose…**. In this state, if you try to save the form, you’d get a standard Moodle form error. The error would appear against the entire ‘Access restrictions’ box rather than on the specific field, because this is actually a single Moodle form control.
  + The **User field** restriction shown here is exactly equivalent to current functionality. As mentioned above, this is only an example of a restriction type.
* There are now two eye icons. The system will show an eye icon for each top-level AND or NOT OR requirement (if the top level choice is OR or NOT AND, you only get a single icon, because it’s not possible to sensibly interpret what to do when OR requirements have a mixture of settings).
  + In this example, you might click the ‘hide’ icon next to the user field restriction so that the activity only appears at all for a certain class of users (e.g. those in a particular department) and other users do not see it at all. However, you could leave the grade item at ‘show’ so that users in the correct department do see the activity, greyed out and with information that they need to obtain the grade.
  + We need this (or similar) functionality so we can reproduce existing Moodle behaviour when using grouping with ‘group members only’, where this entirely hides the item, even where other access restrictions would be shown; also for the current behaviour with ‘available until’ dates. It also adds extra flexibility, such as the example above.

### Negative restrictions

Using the dropdown you can switch to a NOT condition.



* This example means that if a user has grade of at least 70%, or their phone number contains 01908, they will **not** be allowed to access the activity.
* Because the top level condition is NOT ANY, there are still individual hide icons next to each grade. In this case both are visible, so a user might see the activity greyed out with a message like ‘Not available when Phone matches a specified value’, or something similar.

### Nested restrictions

In addition to adding a restriction, the Add restriction… button will also allow you to add a group for restrictions.



This screen (which doesn’t follow on from the previous one) indicates a situation where the ‘outer’ condition is AND type and includes one restriction and two groups. The first group is OR type and contains two restrictions. The second group has just been created and doesn’t have any restrictions yet.

* The nested groups are indicated by a colour change (toggling from grey to white background, in default display) and indent. (Probably the real indent will be more than shown.)
* You can delete a nested group only if it doesn’t contain any restrictions. Deleting the top group in this example would require three clicks (X on the two conditions, then X on the group).
* When initially added, groups just show **None**. The top line **Student must/must not** will appear when you add one restriction to it; the **all/any** choice will appear when you have two or more. This is consistent with the behaviour of the top-level item.
* The visibility icons only appear for top level restrictions; in this case, the grade and the two groups, but not for the other restrictions nested within the groups. This is for simplicity and because it wouldn’t really add functionality.
* In the back end there will be no limit to nesting of groups. The front end might impose a restriction to make styling easier, e.g. no more than 3 levels deep.

## Example use of OR conditions

This example shows real-world use of OR conditions. Assume that we have given students a choice of three projects, and they are assigned to groupings that correspond to these projects. The activity shown here is visible for two of the three projects.



## Accessibility

This version of the interface does not provide drag and drop (which could be added later if required; it’s complicated enough already and you shouldn’t normally need to move things), which means it consists only of basic controls. However, these are dynamically generated and there are some accessibility concerns.

* **Keyboard access**: Needs consideration – the dialog, buttons, icons, and other controls should be keyboard-accessible as normal.
* **High contrast/colour-blind**: OK – no information is indicated solely by colour.
* **Zoom**: Needs consideration – the content of the requirements boxes should wrap if necessary.
* **Screen magnifier**: OK – nothing in the interface is right-aligned.
* **Screen reader**: Needs consideration – standard controls are used, but they are dynamically generated. if possible, suitable ARIA information should be used to indicate dynamic changes. Each field should correctly use associated <label> tag. Icons should have correct title and/or alt text.

## Theme

Required styles for the new interface will be implemented in both ‘legacy’ themes and newer Bootstrap-based themes.

## Backward compatibility

I plan that this system should support all the functionality of the current system (with automatic conversion on upgrade where required), and will have identical results for student usage.

An exception is that there will probably be slight changes in the ‘not available’ text (while restructuring things into separate plugins, I expect the text will be organised slightly differently).

The interface for staff is significantly different, as outlined above, but includes all the functionality from before.

In the code, I intend for public API functions (condition\_info and related) to continue working, although probably with deprecation and name changes as the code is moved to use newer conventions including namespaces.

## Implementation steps

These development tasks are required to implement the feature (roughly in suggested order).

### New data storage

* Define new data field(s).
* Make changes to include this new data in the modinfo cache (or current equivalent) and cm\_info objects. Write unit tests.
* Add a new control to the activity form to provide access to this data field. The control will initially just be a text area in which the developer can manually type JSON data.
* Add backup and restore for the new field. (See also below.) Write unit tests.

**Milestone**: It is now possible to enter data into the new system and it will be stored correctly.

### Restriction API

* Define new plugin type.
* Implement API to use plugins (replacing current availability API). This includes the logic to control access to activities and to display information where access is not granted.
* Do a basic implementation of date restriction as example plugin (so there’s one for testing).
* Implement unit tests.

**Milestone**: The new API works and can implement date restrictions (but nothing else yet).

### Restriction plugins

* Implement plugins equivalent to the options in current availability API (reusing existing code where possible). This includes the following 5 types, shown with their options:
  + Date (from/until a specified time)
  + Grade (selected grade item; more than/less than a specified percentage)
  + User field (specified field, specified text match type e.g. ‘contains’, specified value)
  + Activity completion (specified activity, required completion status)
  + Grouping (selected grouping or none)
* Each plugin should have unit tests.

**Milestone**: The new back-end API now completely works and implements all new features.

### Data upgrade

* Write upgrade script to convert existing availability restrictions into the new data structure.
* The script should take account of the grouping and groupmembersonly settings (and the current special behaviour for ‘available until’ dates). It should turn on enableavailability if required, as mentioned above.
* Script performance should be acceptable (including estimates of how long it will take on OU live data) and it should display progress while operating.

**Milestone**: We can now upgrade from an existing system, and behaviour will be mostly preserved (there will be some issues due to interaction with remaining code that uses the grouping stuff).

### Remove unnecessary data and code

* In an upgrade script, the tables and fields used for current availability restriction (e.g. groupmembersonly) should be removed.
* The code relating to current availability options within the activity form should be removed.
* Based on a search of the code, all remaining usage of these tables/fields should be removed (they are all now handled by the availability API).
* The FEATURE\_ activity setting relating to whether to display grouping (when groups are not supported) should be deprecated and usage removed from core code as this is no longer needed.
* Check existing Moodle unit tests still run and fix any problems caused by the deleted entries.

**Milestone**: Upgrades from existing system should now completely work. Unnecessary stuff should have been completely removed from the user interface and code.

### Restore old backups

* Write code as part of restore so that when you restore from an old backup containing the old format data, it is automatically converted to the new format in a similar manner to the upgrade script above. This will replace the existing restore code for those items.
* Do unit test.

**Milestone**: Users can restore old backups. Back-end changes are now complete.

### Add form user interface

* Write JavaScript code that takes the JSON data from the text area and replaces it with the interface described in this document.
* Add necessary code e.g. form validation that interacts with the JSON data or values set up by the JavaScript.
* Do styling in main and bootstrap themes.

**Milestone**: Feature is complete.