# University of York Use Case - Access Control

## Controlling access to objects with relationships

| | |
|---|---|
| **Title (Goal)** | Controlling access to the repository with relationship driven access control |
| **Primary Actor** | System, Administrators, Cataloguers |
| **Scope** | |
| **Level** | |
| **Story (A paragraph or two describing what happens)** | Each object is covered by licenses to capture the different permissions for that object. These licenses may be standard ones like Creative Commons or may need to be tailored for specific local requirements.<br><br>For example, for read access, we may have a license that says:<br><br>• anyone can see the discovery metadata<br>• only users with a specific role can download the jpeg version of the image<br><br>one that says<br><br>• library staff can create, update and delete this<br><br>and another that says<br><br>• administrators can CRUD everything<br><br>Licenses exist as objects in the repository, related to objects via a hasLicense relationship.<br><br>Access control policies (eg. XACML) are applied to objects indirectly via the license.<br><br>So:<br><br>object <hasLicense> license(s)<br><br>license <hasPolicy> policy(ies)<br><br>The relationship to the license is applied at the object creation stage, or through batch updates by administrators through a management interface.<br><br>In this way, if we need to change the wording of the license statement, we change only one object, or if we need to revise a policy to cover a different, new, datastream, we change only the policy. This also allows us to manage licenses in the repository rather than separately (and as now, on paper!).<br><br>This moves away from hierarchical access control, thereby allowing for flexible collection structures where objects may appear in multiple collections but are secured by their underlying license conditions and not by membership of a collection.<br><br>To deliver truly flexible and efficient access control we also have attribute look-up of parameters, for example to support:<br><br>• module code (restrict read to members enrolled on a specified module of study, as identified by a module code assigned as a role to the logged in user by the authentication process)<br>• username (restrict update to the creator)<br><br>Both of the above would mean a single policy for each, to support many thousands of individual policy cases. |

Note:

This is currently possible as of Fedora 3.5 and above. We have not yet migrated from Fedora 3.4 but are hoping to in 2014. Our main driver for this is to reduce the number of policies in use (less than 10 - 20 policies would probably cover our 100,000 objects rather than the 287 plus we currently have), simplify policy application, improve collection flexibility by removing the collection based application of policies and reduce duplication. The use case is not concerned with getting user roles, rather with how to apply access control to roles that are already defined in the system.