# Item Level Versioning


1 What is Item Level Versioning?
2 Enabling Item Level Versioning
3 Initial Requirements
4 User Interface
    4.1 General behaviour: Linear Versioning
    4.2 Creating a new version of an item
    4.3 View the history and older versions of an item
5 Architecture
    5.1 Versioning model
    5.2 Services to support Versioning and Alternative Identifiers.
        5.2.1 Versioning Service
        5.2.2 Identifier Service
6 Configuration
    6.1 Versioning Service Override
    6.2 Identifier Service Override
    6.3 Version History Visibility
7 Identified Challenges & Known Issues in DSpace 3.0
    7.1 Only Administrators and Collection/Community Administrators can add new versions
    7.2 Conceptual compatibility with Embargo
    7.3 Conceptual compatibility with Item Level Statistics
    7.4 No support for JSP User Interface
    7.5 Exposing version history
8 Credits


## What is Item Level Versioning?

*Versioning* is a new functionality to build the history of an item. Users will have the opportunity to create new version of an existing item any time the will make a change.

## Enabling Item Level Versioning

*By default, Item Level Versioning is disabled in DSpace 3.0*. This feature is currently only supported in the XMLUI.

If you wish to enable this feature, you just have to uncomment the "Versioning" aspect in your `[dspace]/config/xmlui.xconf` file (and restart your servlet container):

```
<!-- ====================
     Item Level Versioning
     ==================== -->
<!-- To enable Item Level Versioning features, uncomment this aspect. -->
<aspect name="Versioning Aspect" path="resource://aspects/Versioning/" />
```

AIP Backup & Restore functionality only works with the Latest Version of Items

If you are using the AIP Backup and Restore functionality to backup / restore / migrate DSpace Content, you must be aware that the "Item Level Versioning" feature is **not yet compatible** with AIP Backup & Restore. **Using them together may result in accidental data loss.** Currently the AIPs that DSpace generates only store the *latest version* of an Item. Therefore, past versions of Items will always be lost when you perform a restore / replace using AIP tools. See DS-1382.

Versioning history exposes data that may be considered personal

If you enable versioning, the **name and email of the submitter are shown to all users** by default in Version history. The only way to circumvent this is to make Version history visible only to admins by setting `item.history.view.admin=false` in `[dspace]/config/modules/versioning.cfg`. Another solution will be prepared, but only after the 3.0 release. See DS-1349.

## Initial Requirements

The Item Level Versioning implementation in DSpace 3.0 builds on following requirements identified by the stakeholders who supported this contribution: Initial Requirements Analysis

1. What should be *Versionable*
    a. Versioning happens at the level of an Individual Item
    b. Versioning should preserve the current state of *metadata*, *bitstreams* and *resource policies* attached to the item.
2. Access, Search and Discovery
    a. Only the most recent version of an item is available via the search interface
    b. Previous versions of Items should continue to be visible, citable and accessible
    c. The Bitstreams for previous versions are retained. If something was once retrievable, it should always be retrievable.
3. Identifiers
    a. Each version of an Item is represented by a separate "*versioned*" identifier
    b. A base "*versionhistory*" Identifier points to the most recent version of the Item.

    c. A revision identifier also exists that is unique to the specific version.
    d. When a new version of an Item is deposited, a new revision identifier will be created.
4. Presentation
    a. On the item page, there is a link to view previous/subsequent versions.
    b. By examining the metadata or identifiers, it is possible to determine whether an item is the most recent version, the original version, or an intermediate version.
5. Access Control and Rights
    a. Certain roles should be able to generate a new version of the item via submission.
    b. To submitters, collection manager, administrators will be given to option to create new version of an item.
    c. Rights to access a specific Item should transmute as well to previous versions
    d. Rights to access a specific Bitstream should also transmute to previous versions.
6. Data Integrity
    a. The relationships between versions should not be brittle and breakable by manipulating Item metadata records.
    b. The relationships between versions should be preserved and predictable in various Metadata Exports (OAI, Packagers, ItemExport)
    c. The relationships between versions should be maintained in SWORD, LNI and AIP packaging and be maintained in updates and restorations.

# User Interface

## General behaviour: Linear Versioning

From the user interface, DSpace offers **linear** versioning. As opposed to hierarchical versioning, linear version has following properties:

- A new version can only be created started from the latest available version
- When new version has been created and still needs to pass certain steps of the workflow, it is temporary impossible to create another new version until the workflow steps are finished and the new version has replaced the previous one.

## Creating a new version of an item

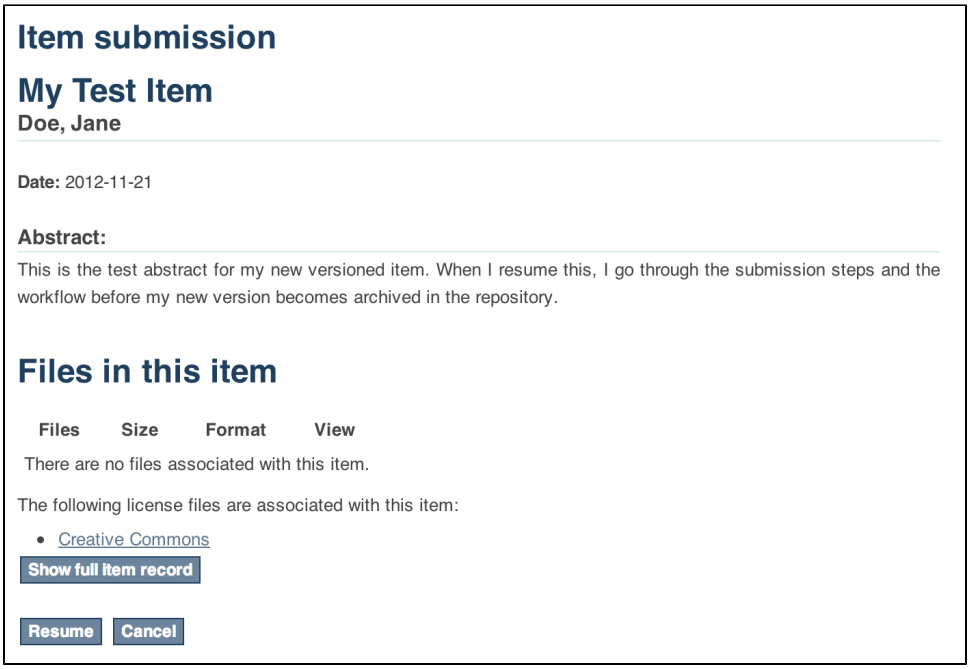
Administrators and collection/community administrators can create new versions of an item from the Item View page.

1. Click "Create a new version" from the Context Menu in the navigation bar.
2. Provide the reason for creating a new version that will lateron be stored and displayed in the version summary.




3. Your new version is now creates as a new Item in your Workspace. It requires you to go through the submission and workflow steps like you would do for a normal, new submission to the collection. The rationale behind this is that if you are adding new files or metadata, you will also need to accept the license for them. In addition to this, the versioning functionality does not bypass any quality control embedded in the workflow

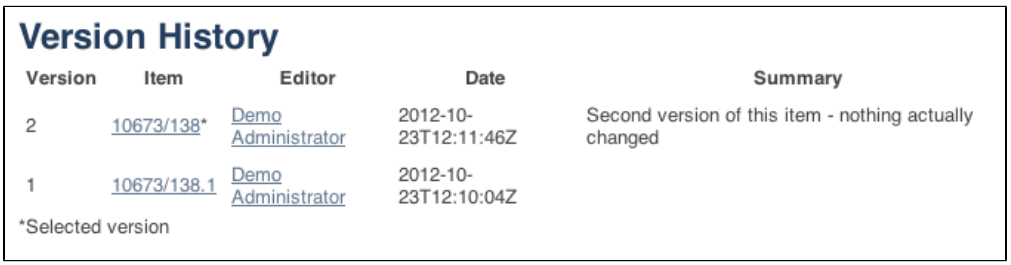
steps.




After the submission steps and the execution of subsequent workflow steps, the new version becomes available in the repository.

### View the history and older versions of an item

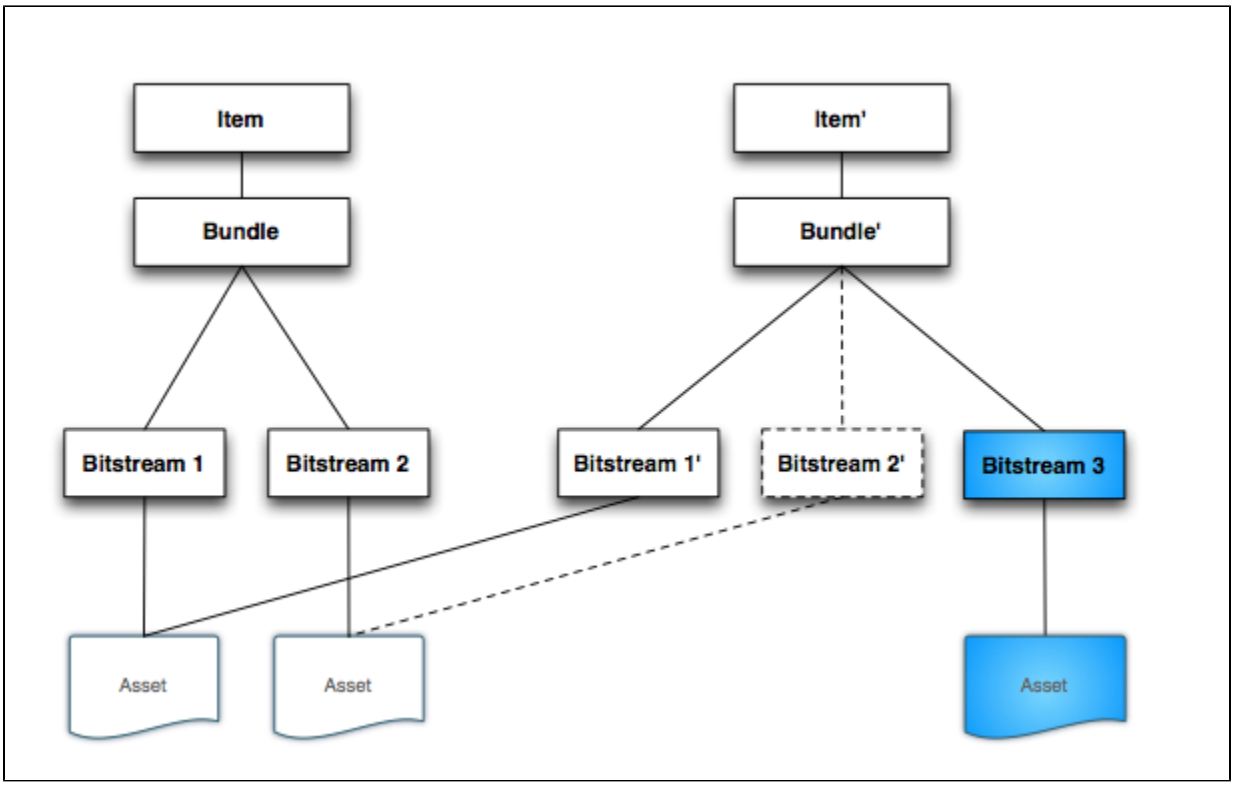
An overview of the version history, including links to older versions of an item, is available at the bottom of an Item View page. The repository administrator can decide whether the version history should be available to all users or restricted to administrators.




## Architecture

### Versioning model

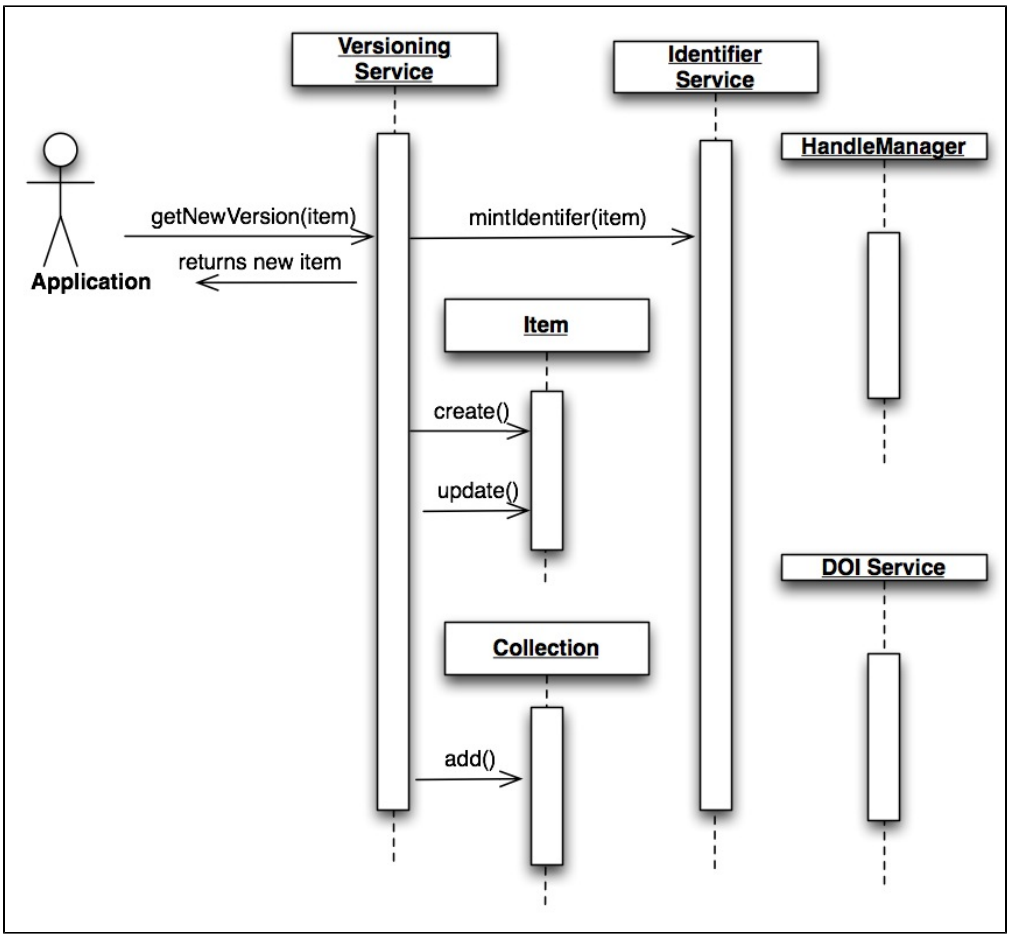
For every new Version a separate DSpace Item will be created that replicates the metadata, bundle and bitstream records. The bitstream records will point to the same file on the disk.




The *Cleanup* method has been modified to retain the file if another Bitstream record point to it (the dotted lines in the diagram represent a bitstream deleted in the new version), in other words the file will be deleted only if the Bitstream record processed is the only one to point to the file *(count(INTERNAL_ID)=1).*

## Services to support Versioning and Alternative Identifiers.

DSpace Item Versioning will be encapsulated as an Extensible Service that may be reimplemented by the local repository maintainers to produce alternate versioning behaviors and Identifier Schemes. Versioning Services layer on top of IdentifierServices dedicated to Encoding, Resolution, Minting and Registration of Identifiers for specific DSpace Items and Bitstreams. It is through this highly extensible layering of functionality where local developers can alter the versioning behavior and introduce their own local enhancements.  The DSpace Service Manager, based on the Spring Framework, provides the key leverage for this flexibility.




### Versioning Service

The *Versioning* Service will be responsible for the replication of one or more Items when a new version is requested. The new version will not yet be preserved in the Repository, it will be preserved when the databases transactional window is completed, thus when errors arise in the *versioning* process, the database will be properly kept in its original state and the application will alert that an exception has occurred that is in need of correction.

The *Versioning* Service will rely on a generic *IdentifierService* that is described below for minting and registering any identifiers that are required to track the revision history of the Items.

```
public interface VersioningService {

    Version createNewVersion(Context c, int itemId);

    Version createNewVersion(Context c, int itemId, String summary);

    void removeVersion(Context c, int versionID);

    void removeVersion(Context c, Item item);

    Version getVersion(Context c, int versionID);

    Version restoreVersion(Context c, int versionID);

    Version restoreVersion(Context c, int versionID, String summary);

    VersionHistory findVersionHistory(Context c, int itemId);

    Version updateVersion(Context c, int itemId, String summary);

    Version getVersion(Context c, Item item);
}
```

### Identifier Service

The Identifier Service maintains an extensible set of *IdentifierProvider* services that are responsible for two important activities in Identifier management:

1. Resolution: *IdentifierService* act in a manner similar to the existing HandleManager in DSpace, allowing for resolution of DSpace Items from provided identifiers.
2. Minting: Minting is the act of reserving and returning an identifier that may be used with a specific DSpaceObject.
3. Registering: Registering is the act of recording the existence of a minted identifier with an external persistent resolver service, these services may reside on the local machine (HandleManager) or exist as external services (PURL or DEZID DOI registrations services)

```
public interface IdentifierService {

    /**
     *
     * @param context
     * @param dso
     * @param identifier
     * @return
     */
    String lookup(Context context, DSpaceObject dso, Class<? extends Identifier> identifier);

    /**
     *
     * This will resolve a DSpaceObject based on a provided Identifier.  The Service will interrogate
the providers in
     * no particular order and return the first successful result discovered.  If no resolution is
successful,
     * the method will return null if no object is found.
     *
     * TODO: Verify null is returned.
     *
     * @param context
     * @param identifier
     * @return
     * @throws IdentifierNotFoundException
     * @throws IdentifierNotResolvableException
     */
    DSpaceObject resolve(Context context, String identifier) throws IdentifierNotFoundException,
IdentifierNotResolvableException;

    /**
     *
     * Reserves any identifiers necessary based on the capabilities of all providers in the service.
     *
     * @param context
     * @param dso
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void reserve(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;

    /**
     *
     * Used to Reserve a Specific Identifier (for example a Handle,  hdl:1234.5/6) The provider is
responsible for
     * Detecting and Processing the appropriate identifier, all Providers are interrogated, multiple
providers
     * can process the same identifier.
     *
     * @param context
     * @param dso
     * @param identifier
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void reserve(Context context, DSpaceObject dso, String identifier) throws AuthorizeException,
SQLException, IdentifierException;
```

```
    /**
     *
     * @param context
     * @param dso
     * @return
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void register(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;

    /**
     *
     * Used to Register a Specific Identifier (for example a Handle,  hdl:1234.5/6) The provider is
responsible for
     * Detecting and Processing the appropriate identifier, all Providers are interrogated, multiple
providers
     * can process the same identifier.
     *
     * @param context
     * @param dso
     * @param identifier
     * @return
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void register(Context context, DSpaceObject dso, String identifier) throws AuthorizeException,
SQLException, IdentifierException;

    /**
     * Delete (Unbind) all identifiers registered for a specific DSpace item. Identifiers are "unbound"
across
     * all providers in no particular order.
     *
     * @param context
     * @param dso
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void delete(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;

    /**
     * Used to Delete a Specific Identifier (for example a Handle,  hdl:1234.5/6) The provider is
responsible for
     * Detecting and Processing the appropriate identifier, all Providers are interrogated, multiple
providers
     * can process the same identifier.
     *
     * @param context
     * @param dso
     * @param identifier
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void delete(Context context, DSpaceObject dso, String identifier) throws AuthorizeException,
SQLException, IdentifierException;

}
```

## Configuration

### Versioning Service Override

You can override the default behaviour of the Versioning Service using following XML configuration file, deployed under your dspace installation directory:

[dspace_installation_dir]/config/spring/api/versioning-service.xml

In this file, you can specify which metadata fields are automatically "reset" (i.e. cleared out) during the creation of a new item version. By default, all metadata values (and bitstreams) are copied over to the newly created version, with the exception of **dc.date.accessioned** and **dc.description.provenance.** You may specify additional metadata fields to reset by adding them to the "ignoredMetadataFields" property in the "versioning-service.xml" file:

```
<!-- Default Item Versioning Provider, defines behavior for replicating
     Item, Metadata, Budles and Bitstreams. Autowired at this time. -->
<bean class="org.dspace.versioning.DefaultItemVersionProvider">
    <property name="ignoredMetadataFields">
         <set>
            <value>dc.date.accessioned</value>
            <value>dc.description.provenance</value>
         </set>
    </property>
</bean>
```

### Identifier Service Override

You can override the default behaviour of the Identifier Service using following XML configuration file, deployed under your dspace installation directory:

[dspace_installation_dir]/config/spring/api/identifier-service.xml

No changes to this file are required to enable Versioning. This file is currently only relevant if you aim to develop your own implementation of versioning.

### Version History Visibility



Version History

| Version | Item | Editor | Date | Summary |
|---|---|---|---|---|
| 2 | 10673/138* | Demo Administrator | 2012-10-23T12:11:46Z | Second version of this item - nothing actually changed |
| 1 | 10673/138.1 | Demo Administrator | 2012-10-23T12:10:04Z | |

*Selected version

By default, **all** users will be able to see the version history. To ensure that only administrators can see the Version History, enable item.history.view.admin in following configuration file:

[dspace_installation_dir]/config/modules/versioning.cfg

```
item.history.view.admin=false
```

## Identified Challenges & Known Issues in DSpace 3.0

Item Level Versioning has a substantial conceptual impact on many DSpace features. Therefore it has been accepted into DSpace 3.0 as an optional feature. Following challenges have been identified in the current implementation. As an early adopter of the Item Level Versioning feature, your input is greatly appreciated on any of these.

### Only Administrators and Collection/Community Administrators can add new versions

There is currently no configuration option to allow submitters to create new versions of an item. This functionality is restricted to Administrators and Collection/Community Administrators. In a context where original submission of DSpace items is done by non-administrator users, it might also make sense to allow them to create new versions. Especially given the fact that new versions have to pass through the workflow anyway.

### Conceptual compatibility with Embargo

Lifting an embargo currently does not interact with Item Level Versioning. Additional implementation would be required to ensure that lifting an embargo actually creates a new version of the item.

## Conceptual compatibility with Item Level Statistics

Both on the level of pageviews and downloads, different versions of an item are treated as different items. As a result, an end user will have the impression that the stats are being "reset" once a new version is installed, because the previous downloads and pageviews are allocated to the previous version.

One possible solution would be to present an end user with aggregated statistics across all viewers, and give administrators the possibility to view statistics per version.

## No support for JSP User Interface

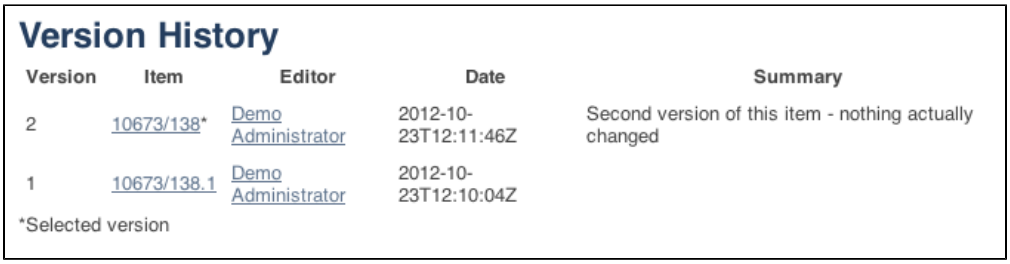
Item Versioning in DSpace 3.0 was contributed without specific user interface support for JSP User Interface.

## Exposing version history

The version history is added on the bottom of a versioned item page. A repository administrator can either decide to show this to all users, or restrict it to admins only. If it is shown to admins only, an end user will have no way to find the way to an older version. On the other hand, if a repository administrator does decide to expose version history to all users, the name and email address of the editor are exposed as well. This might actually be useful if the editor account is a generic institutional email address, but may conflict with local privacy laws if any personal details are included in this account.

Therefore, discussion has illustrated that there is a usecase for an intermediate exposure of version history that hides the Editor column.

You can join the discussion or contribute a new code here:
JIRA DS-1349 - Item Level Versioning exposes personal data



**Version History**

| Version | Item | Editor | Date | Summary |
|---|---|---|---|---|
| 2 | 10673/138* | Demo Administrator | 2012-10-23T12:11:46Z | Second version of this item - nothing actually changed |
| 1 | 10673/138.1 | Demo Administrator | 2012-10-23T12:10:04Z | |

*Selected version

# Credits

The initial contribution of Item Level Versioning to DSpace 3.0 was implemented by @mire with kind support from:

- MBLWHOI Library
- Woods Hole Oceanographic Institution
- Marine Biology Laboratory, Center for Library and Informatics, History and Philosophy of Science program
- Arizona State University, Center for Biology and Society
- Dryad