1 The Repository Model

1.1 Compliance Levels

This specification is divided into two compliance levels and a set of additional optional features which repositories of either level may support. Level 1 provides for read functions and level 2 adds additional write functions. The functional division is as follows:

Level 1 includes:

- Retrieval and traversal of nodes and properties
- Reading the values of properties
- Transient namespace remapping
- Export to XML/SAX
- Query facility with XPath syntax
- Discovery of available node types
- Discovery of access control permissions

Level 2 adds:

- Adding and removing nodes and properties
- Writing the values of properties
- Persistent namespace changes
- Import from XML/SAX
- Assigning node types to nodes

Optional:

Any combination of the following features may be added to an implementation of either level.

- Transactions
- Basic Versioning to support a linear version model
- Full Versioning to support branching, merging, baselines and activities
- Observation (Events)
- Locking
- SQL syntax for query

Formatted: Bullets and Numbering

1.2 Workspaces

A content repository is composed of a number of *workspaces*. Each workspace contains a single rooted tree of items. In the simplest case a repository will consist of just one workspace. In more complex cases a repository will consist of more than one workspace.

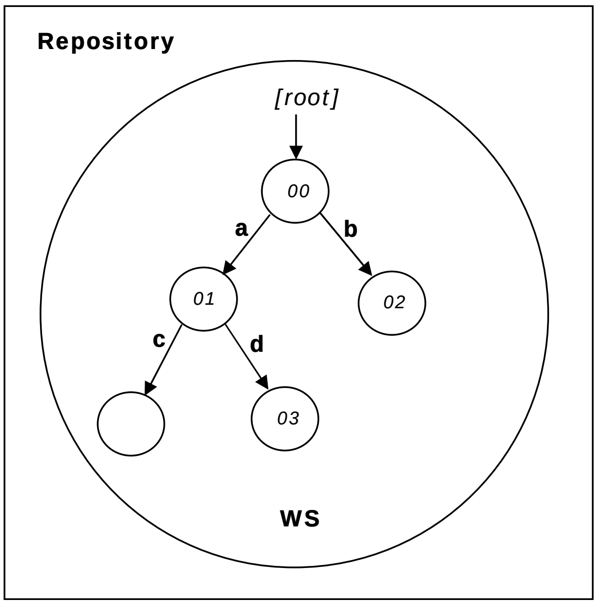
Workspaces are the entities into which applications and users login. Workspaces are also the objects responsible for query and define the scope of the queries that they execute. Therefore, care should be taken to define workspaces to include the scope in which applications need to query.

1.2.1 Single Workspace Repositories

A repository with only a single workspace consists of a single tree of nodes and properties. The example at the beginning of this section (4 *The Repository Model*) describes a single workspace repository.

Since a given workspace contains at most one node with a given UUID, in this case, there is at most one node with a given UUID *in the repository as a whole*.

The following diagram depicts a single workspace repository:



The small circles represent nodes. The arrows point from parent to child and are labeled with the name of the child. The name of the root node is actually the empty string though, for clarity, it is indicated here with the string "*[root]*". The numbers within the nodes represent the UUIDs of the nodes. For example, the UUID of the root node **/** is *00* and the UUID of **/a/d** is *03*. The node **/a/c** is not referenceable and therefore does not have a UUID.

1.2.2 Workspaces and Basic Versioning

Formatted: Bullets and Numbering

Formatted: Normal

In repositories that support basic versioning, a repository is assumed to have a single operational workspace and a version workspace for querying the version store. The operational store is used for most update operations and for general querying of current content.

A workspace identifier can be obtained from the repository for the version store that presents all of the versions of all nodes in the repository. This workspace is read-only to query the versions and version histories in the version store. As such, multiple workspace operations defined in the next section are not supported.

1.2.3 Multiple Workspaces and Corresponding Nodes

Formatted: Bullets and Numbering

Any repository can potentially support multiple workspaces. In implementations that support the full versioning system, a repository can have multiple workspaces other than the version workspace that can share corresponding changes between workspaces. In these repositories, a node in one workspace may have *corresponding nodes* in other workspaces. A node's corresponding node is defined as follows:

Deleted: repositories

Deleted: In repositories that have multiple workspaces,

- A node's corresponding nodes are those with the same *correspondence identifier*.
- The correspondence identifier of a referenceable node is its UUID.
- The correspondence identifier of a non-referenceable node with at least one referenceable ancestor is the pair consisting of the UUID of its nearest referenceable ancestor and its relative path from that ancestor.
- The correspondence identifier of a non-referenceable node with no referenceable ancestor is its absolute path.

Recall also that (as stated in **Error! Reference source not found. Error! Reference source not found.**) if a repository has a workspace with a referenceable root node then *all* its workspaces must have referenceable root nodes *and* those root nodes must all have the same UUID.

Apart from having the same correspondence identifier, corresponding nodes need have nothing else in common. They can have entirely different properties and child nodes, for example.

While a node *may* have a corresponding node in another workspace, it is not *required* to.

Note that there is still at most one node with a given UUID per workspace.

The **`update`** method,

`Node.update(String srcWorkspace)`

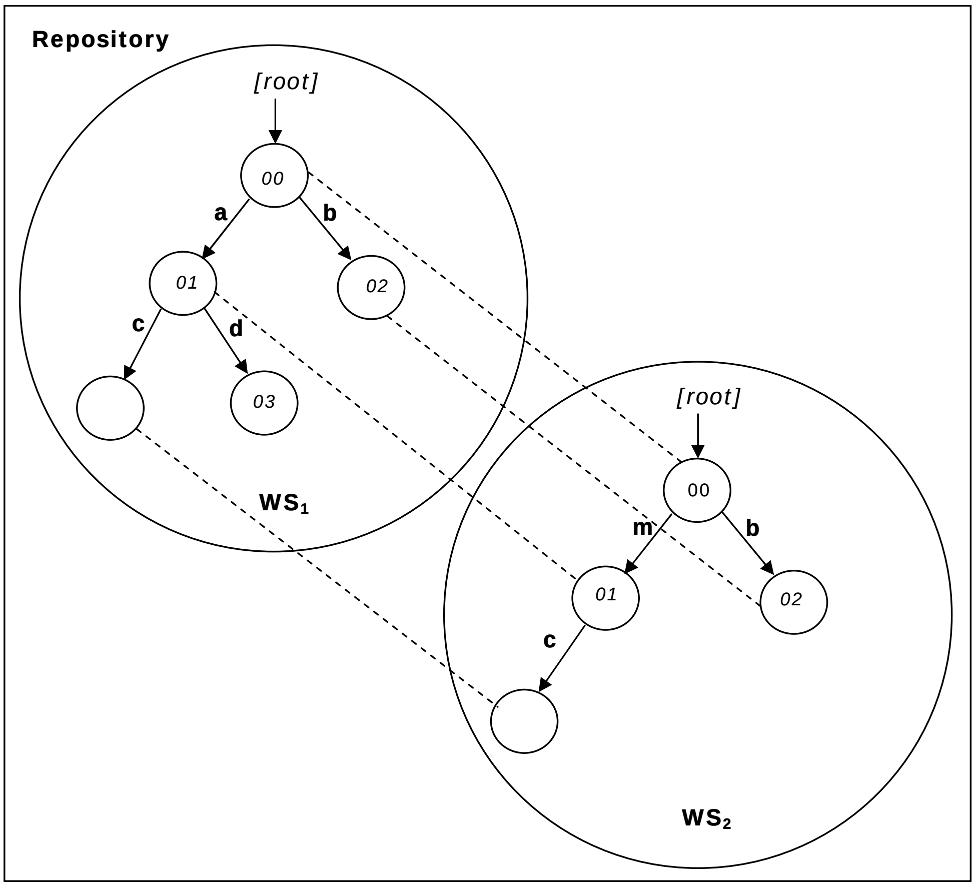
causes **`this`** node and its subtree to be replaced by a clone of this nodes corresponding node and its subtree in **`srcWorkspace`**.

For more details on corresponding nodes and the update method see 7.1.8 *Updating and Cloning Nodes across Workspaces*.

1.2.3.1 Example

Formatted: Bullets and Numbering

The following diagram shows a schematic of a two-workspace repository.



Here we see two workspaces, **WS₁** and **WS₂**. The dotted lines indicate corresponding nodes. For example, the node **/a** in **WS₁** corresponds to **/m** in **WS₂** because both have a UUID of *01.* Similarly, **/b** in **WS1** corresponds with **/b** in **WS2**. In these cases, because the nodes are referenceable, their paths and names are not relevant in determining their correspondence.

On the other hand, **/a/c** in **WS₁** corresponds with **/m/c** in **WS₂** because they have the same relative path (namely, **c**) from their nearest referenceable corresponding ancestors (namely, **/a** and **/m** in **WS₁** and **WS₂** respectively).

Note there can also be nodes (such as **/a/d** in **WS₁**) that exist in one workspace but not in the other.

1.3 Versioning

Support for versioning is an optional feature. The versioning system is built on top of the system of workspaces and referenceable nodes described above. There are two levels of versioning available - a basic versioning model for supporting versioning common to most content management systems and a full versioning model to support branching and merging between multiple workspaces and more advanced features such as baselines and activities.

In a repository that supports versioning, either with the basic versioning model or full versioning model, a workspace may contain both *versionable* and *nonversionable* nodes. A node is versionable if and only if it has been assigned the *mixin type* **`mix:versionable`**, otherwise it is nonversionable. Repositories that do not support versioning will simply not provide this mixin type, whereas repositories that do support versioning must provide it. The type **`mix:versionable`** is a subtype of **`mix:referenceable`**, so if a node is versionable it is automatically also referenceable and thus has a UUID.

Being versionable means that at any given time the node's state can be saved for possible future recovery. This saved state is called a *version* and the action of saving it is called *checking in*.

Versions exist as part of a *version history*. Version histories contain all previous versions of the node. Version histories are accessible from any node associated with that version history. In addition, all the versions of a node can be accessed from the version history.

The basic versioning model provides a simple model of version control. Version histories provide a linear list of versions ordered in chronological order of creation date. Versions are limited to a single workspace for update operations on nodes. No branching or merging of versions is supported.

Deleted: active versions of

The full versioning model provides a more sophisticated model of versioning somewhat similar to those found in source code control systems. In this model within a version history, the versions form a *version graph* that describes the predecessor/successor relations among versions of a particular versionable node. These version graphs can be shared between multiple workspaces to allow concurrent update and development of nodes with interfaces to support merging the changes between workspaces.

Deleted: Within

Version histories and their contained versions are stored in version storage. There is one version storage per repository. This version storage is accessed through a separate version workspace for querying all versions of all nodes. In the full versioning model, the version store may be exposed in each workspace as a special protected subtree below the node

Deleted: , though it is

/jcr:system/jcr:versionStorage for backward compatibility with JSR-170.

Deleted: .

1.3.1.1 Relation Between Nodes and Version Histories in the Full Versioning Model

In the full versioning model, the relationship between nodes and version histories is built on the notion of correspondence via UUID. The details are as follows:

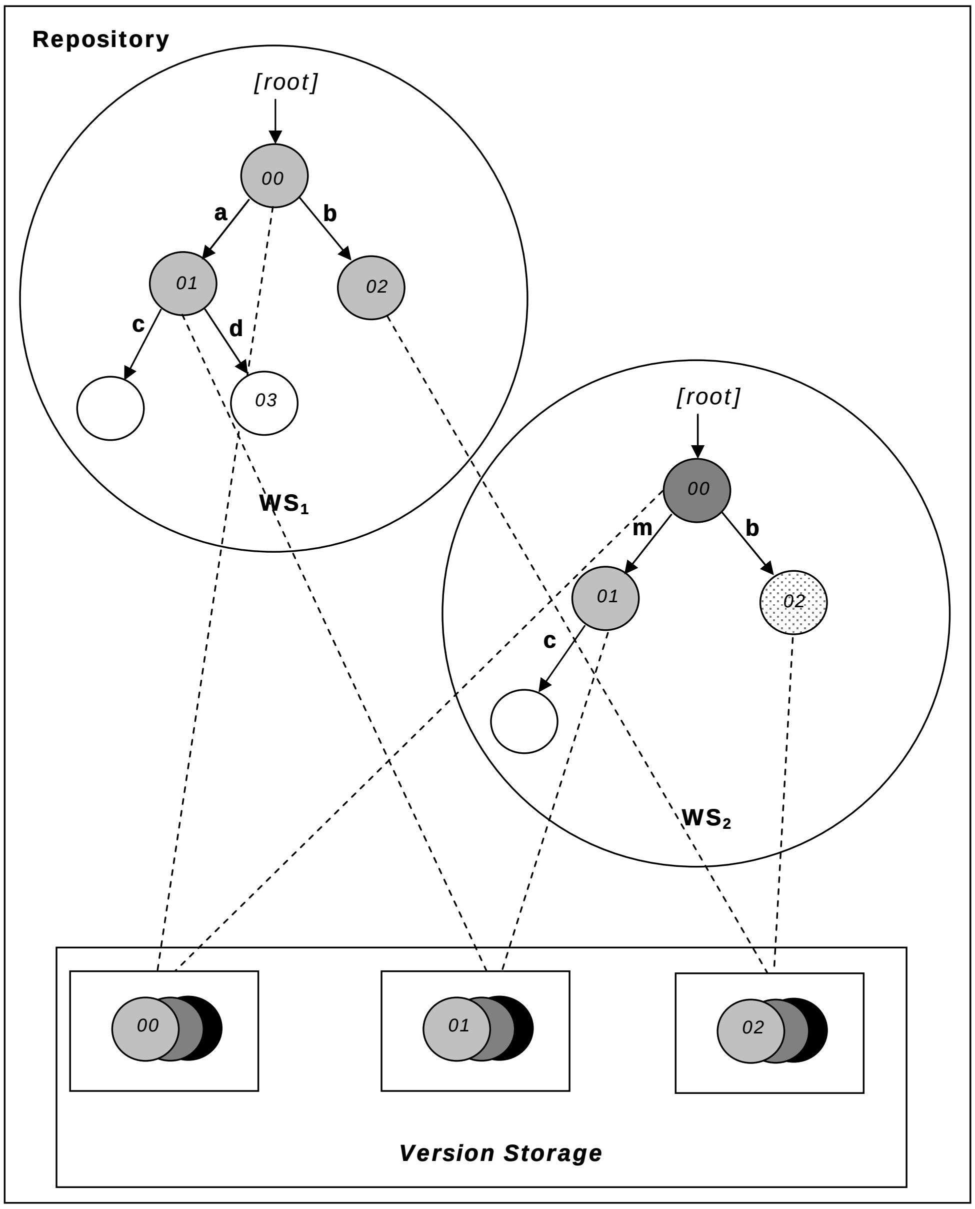
Deleted: The

- Each set of corresponding versionable nodes (nodes with the same UUID) share the same version history.
- In a given workspace, there is at most one versionable node per version history (this follows directly from the fact that there is at most one node from each correspondence set per workspace).
- Given a particular workspace, there may be version histories for which that particular workspace does not contain a corresponding versionable node.
- A workspace may contain nonversionable nodes, which, of course, never have corresponding version histories.
- When a new versionable node is created (i.e., the first instance in the repository as whole) a version history for that node is automatically created in version storage.
- If a versionable node is cloned to another workspace, it maintains the same UUID and the new corresponding versionable node remains associated with the original's version history.
- Note that since all versionable nodes are by definition referenceable, there is no need to include the qualification involving relative paths to the nearest versionable node (or root node) as in the discussion of **update**, above.

Formatted: Page break before

1.3.1.2 Example

The following diagram illustrates a possible repository architecture.



This diagram shows a repository that supports versioning and contains two workspaces. The version storage is represented by the area in the bottom. It contains a version history for each versionable node in the repository. The versionable nodes in the workspaces are shown in various shadings. The nonversionable nodes are shown in white.

All versionable nodes are referenceable, though not all referenceable nodes are versionable (for example the node *03* in **WS$_1$** is referenceable, because it has a UUID, but it is not versionable). Both **WS$_1$** and **WS$_2$** also contain nonreferenceable nodes (the nodes **c** below *01*).

In the diagram the version histories are represented by stacked circles of differing shades. Each versionable node shares its version history with its corresponding node in the other workspace.

At any given time a particular workspace may hold nodes based on various versions stored in version storage. In the diagram, **WS$_1$** holds nodes based on the “light gray” version of the nodes *00*, *01* and *02*. **WS$_2$**, in contrast, has nodes based on the “dark gray” version of *00*, the “light gray” version of *01* and the “dotted” version of *02*.

Note that for the purposes of illustration, each version history is depicted as containing three versions. This is a simplification; in an actual system the version histories of distinct nodes may differ. Furthermore, in this picture, parent child relations within the version storage are not shown. See 8.2 *Versioning* for a more detailed description.