

# JSR 170 Content Repositories

---

Hermod Opstvedt  
Chief Architect  
DnB NOR ITUD





# JSR 170

---

- What is a content repository?
  - A content repository is that facility that contains the content managed, displayed or just used by a content application such as a content management system (CMS).



# JSR 170

---

- A Content Repository can use a variety of different data sources for the storage of the content such as a file system, a relational database, or an OO database designed for media assets and large multimedia objects and all the metadata associated with such objects.



# JSR 170

---

- The set of functionality defined for a "content repository" includes:
  - Dealing with structured and unstructured content.
  - Read and write capabilities.
  - BLOB (multimedia), text and other data types.
  - Fine grained and coarse grained.



# JSR 170

---

- Hierarchy operations, parent/child relationship query mechanisms.
- Referential integrity.
- Versioning operations.
- Observation.
- Transactional capabilities.
- Locking.
- Serialization (export/import).
- Access control.



# JSR 170

---

- History

- Original work by Day Software.
- Submitted to JSR 2002.
- Specification lead: David Nuescheler, Day Software.
- First final release 17 June 2005.
- Current version 1.0.1, released 22 mar 2006.



# JSR 170

---

- Why JSR 170?
  - Due to an increase in the number of vendors offering proprietary content repositories, a need for a common API arose.
  - A common API would facilitate an industry wide content infrastructure.



# JSR 170

---

- The Repository Model:
  - One ore more Workspaces.
  - Contains a tree of Items.
    - May be a node or a property.
  - All Items except the root have Parents.
  - Property Items can not have children.
  - All Items except root have non-empty names.



# JSR 170

---

- Root Item always has an empty String as its name.
- A property and node with the same parent cannot have the same name.
- More than one node with the same parent may have the same name, but are distinguished by index.

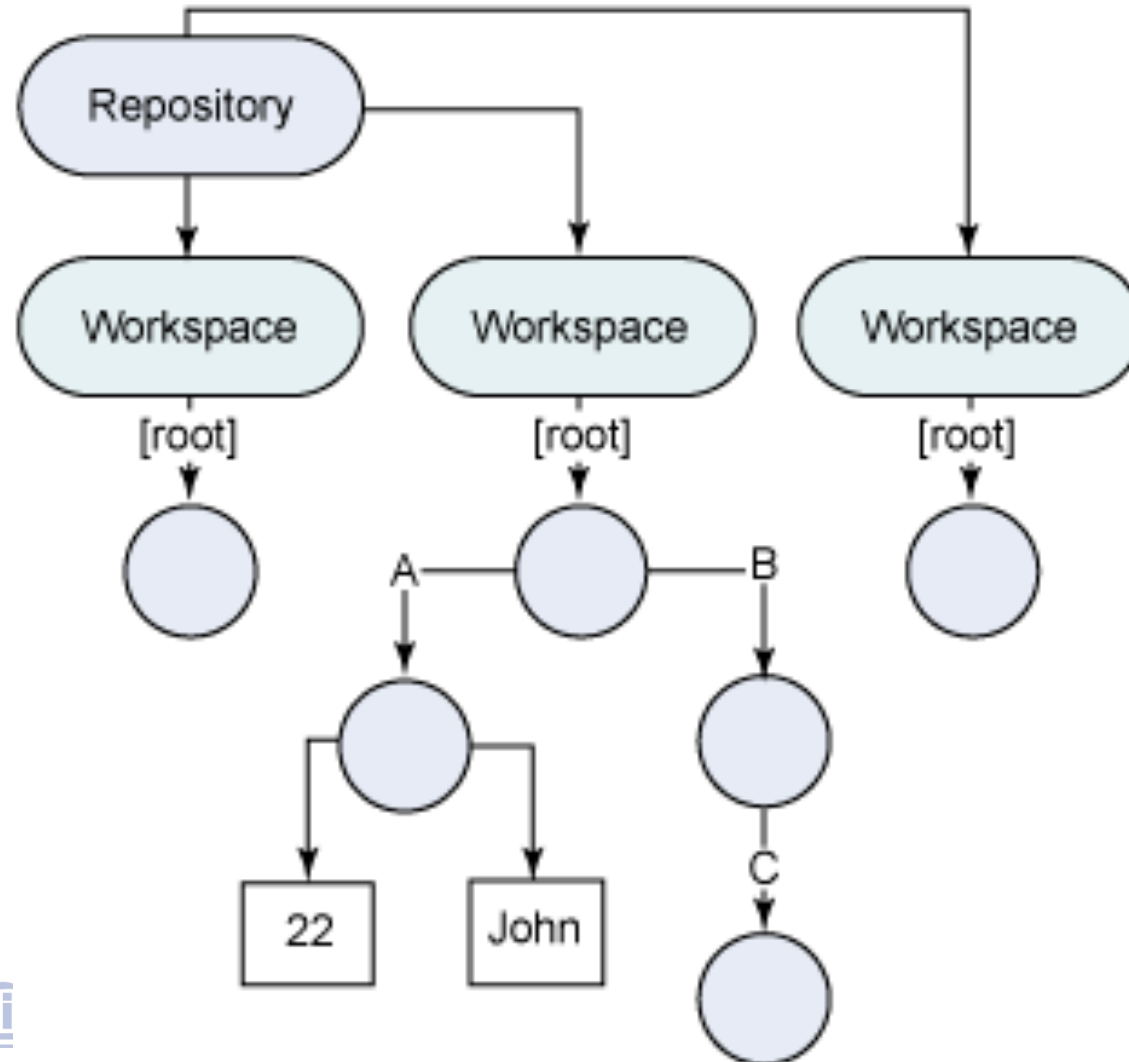


# JSR 170

---

- A node is identified by its chain of names in the same way as a directory path on a file system.
  - Ex. / is the root node
  - Ex a/b/c/d identifies the node d.
- A node may be referred to by its absolute path or by a relative path.
- Absolute paths always start with /.

# JSR 170





# JSR 170

---

- No need for programmers and solution providers to learn a new API each time a new repository vendor comes along.
- The ability to change the underlying content repository with out having to change the CMS that manages it.



# JSR 170

---

- **Goals:**
  - It should not be tied to any particular underlying architecture, data source or protocol.
  - It should be easy to use from the programmer's point of view.



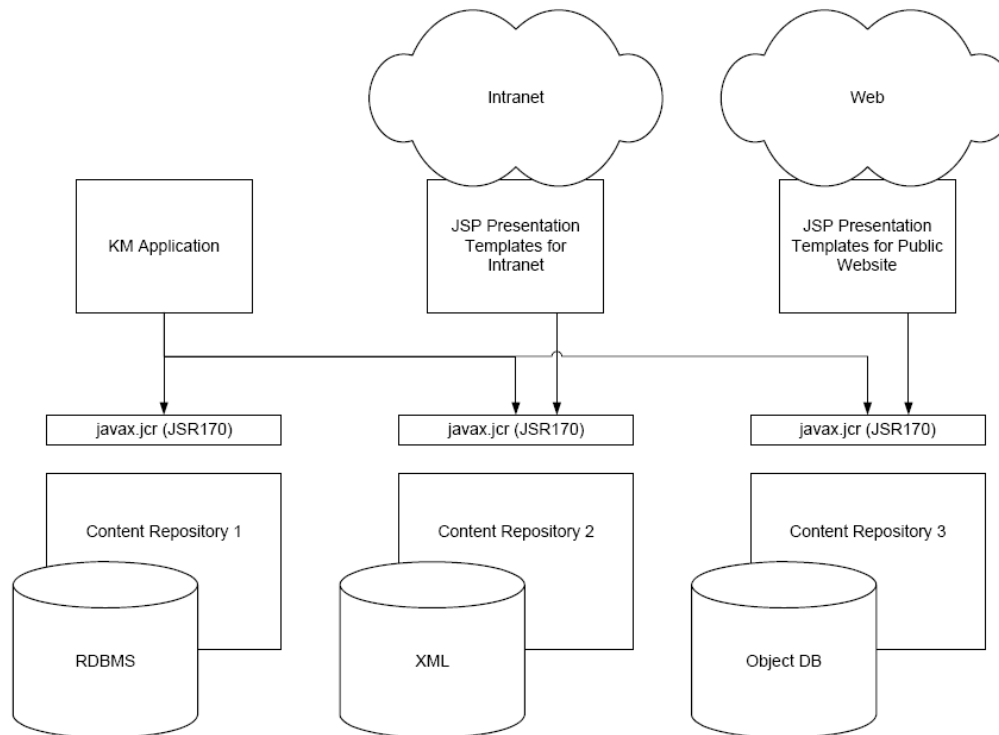
# JSR 170

---

- It should allow for relatively easy implementation on top of as wide a variety of existing content repositories as possible.
- It should standardize more complex functionality needed by advanced content-related applications.

# JSR 170

- A typical scenario:





# JSR 170

---

- The API:
  - Repository (`javax.jcr.Repository`)
    - Is the base of the API
    - A user logs into the Repository
  - The specification does not deal with how one acquires a Repository.
    - JNDI lookups
    - Direct instantiation of implementations
    - Factory methods.



# JSR 170

---

- Methods on repository:
  - `getDescriptor(String key)` – returns `String`
  - `getDescriptorKeys` – returns `String[]`
  - `login` – returns `Session`
    - `()` – returns `Session`
    - `(Credentials credentials)`
    - `(Credentials credentials, String workspaceName)`
    - `(String workspaceName)`



# JSR 170

---

- Credentials (`javax.jcr.Credentials`)
  - Is used to build an authentication object that one uses to log into the repository workspace.
- Session (`javax.jcr.Session`)
  - The Session object provides read and write access to the content of a particular workspace in the repository.



# JCR 170

---

- Item (javax.jcr.Item)
  - The Item is the base interface of Node and Property.
- Node (javax.jcr.Node)
  - The Node interface represents a node in the hierarchy that makes up the repository.



# JCR 170

---

- Primary vs Mixin
  - There are two categories of node types, primary and mixin. Every node has a primary node type assigned to it upon creation in addition, a mixin node type may be added to a node later in its lifecycle.



# JCR 170

---

- The primary node type of a node usually defines node structure (*i.e.*, allowed and required child nodes and properties) related to the problem domain being modeled. For example, a node used in storing content about business contacts might have the primary type `myapp:Contact` which defines properties such as `myapp:givenName`, `myapp:familyName` and so forth.



# JCR 170

---

- Mixin node types usually specify additional properties or child nodes related to a capability being added to the node. These capabilities may include generic repository-level functions as in the case of the built-in mixins `mix:versionable` and `mix:lockable`, for example, or domain-level capabilities such as a `myapp:Emailable` mixin type that adds the property `myapp:emailAddress` to a node.



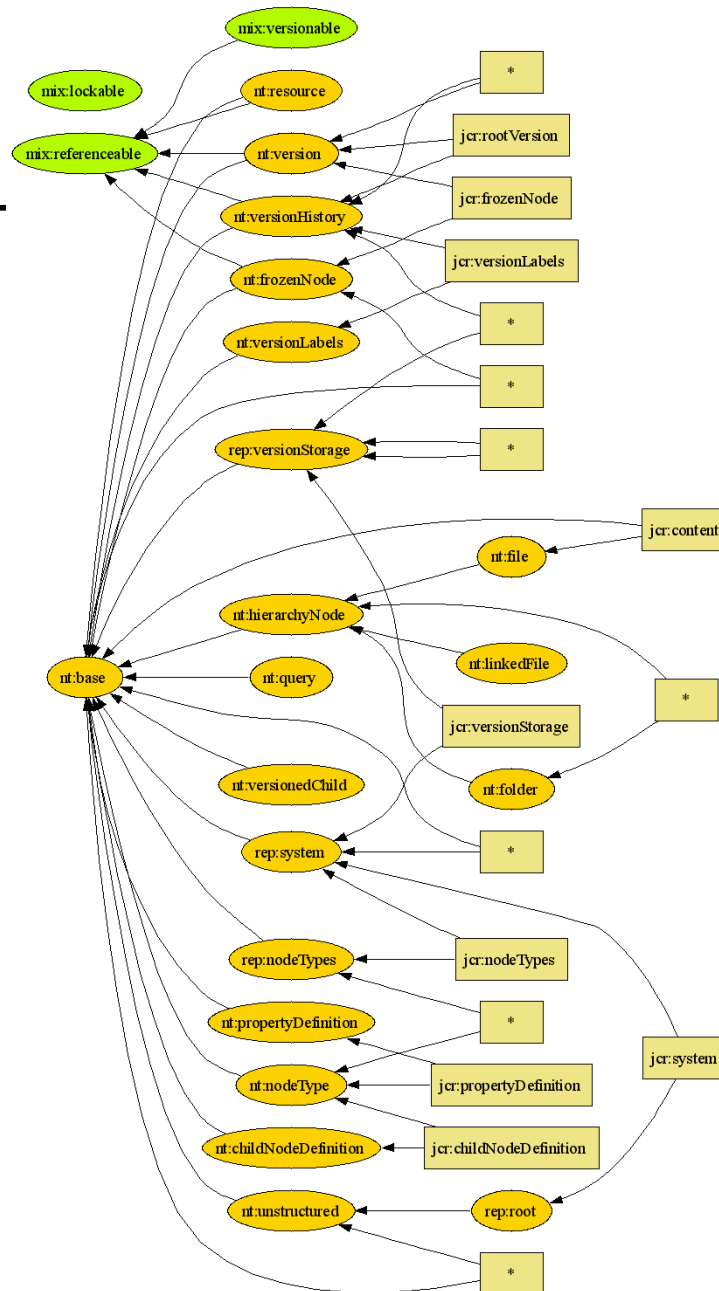
# JSR 170

---

- Property (`javax.jcr.Property`)
  - A Property object represents the smallest granularity of content storage.
  - A property must have one and only one parent node.
  - A property does not have children.
  - A property consists of a name and a value.

# JSR 170

- Node types





# JSR 170

---

- Find data:
  - Using simple traversal access.
  - Using XPath.
  - Using SQL Query.



# JSR 170

---

- Using XPath

- XPath is a language for finding information in an XML document. XPath is used to navigate through elements and attributes in an XML document.
- Current support in JSR 170 is XPath 1.0.
- JSR 170 does not support the full syntax of XPath 1.0.



# JSR 170

---

- Simple traversal access
  - Specify the search using straight forward directory syntax
    - Either relative – a/b/c, ../b/c, *etc.*
    - Or absolute - /a/b/c
  - Use UUID's
    - getUUID
    - Only works on Nodes with node-type:
      - ✓ mix:referenceable



# JSR 170

---

## ➤ XPath samples

- //
  - ✓ Returns the root node
- //element(\*, nt:unstructured)/Node1
  - ✓ Selects root node siblings with a name of Node1
- //element(\*, mynt:paragraph)[@myapp:title="Node Types"]
  - ✓ Returns all attributes that have a title of 'Node Types', and whose parent's nodetype is of type paragraph.



# JSR 170

---

- Using SQL Query
  - Not required part of the JSR 170 spec.
  - Very similar to standard SQL.
  - Well suited where the underlying implementation of the Content Repository is a relational database.



# JSR 170

---

- SQL query samples
  - Select \* from nt:unstructured
    - Returns all nodes of type unstructured (default node type)
  - Select \* from nt:unstructured where title='CSS 2007'
    - Returns all nodes with attributes with title equals CSS 2007.



# JSR 170

---

- XPath – SQL Query comparison

XPath	SQL Query
<code>//*</code>	<code>SELECT * FROM nt:base</code>
<code>//element(*, my:type)</code>	<code>SELECT * FROM my:type</code>
<code>//element(*, my:type)/@my:title</code>	<code>SELECT my:title FROM my:type</code>
<code>//element(*, my:type)/ (@my:title   @my:text)</code>	<code>SELECT my:title, my:text FROM my:type</code>



# JSR 170

---

- Versioning

- Support for versioning is an optional feature in JSR 170.
- Nodes must be assigned the mixin type `mix:versionable`.
  - `mix:versionable` is a subtype of `mix:referenceable`
- Means that at any given time the node's state can be saved for future recovery.



# JSR 170

---

- Part of a version history.
- The versions form a version graph.
- Version histories and their contained versions are stored in version storage.
- One version storage per repository.
- The process of versioning is known as checkin.

The logo graphic consists of a vertical black line intersecting a horizontal black line. To the left of the intersection, there are three overlapping squares: a yellow one at the top, a red one in the middle, and a blue one at the bottom. The word "Jackrabbit" is written in a blue, sans-serif font to the right of the graphic.

# Jackrabbit

---

- What is Jackrabbit?
  - A fully conforming implementation of the Content Repository for Java Technology API.
  - Initial implementation by Day Software.
  - Official reference implementation and technology compatibility kit released along with the final JCR API.

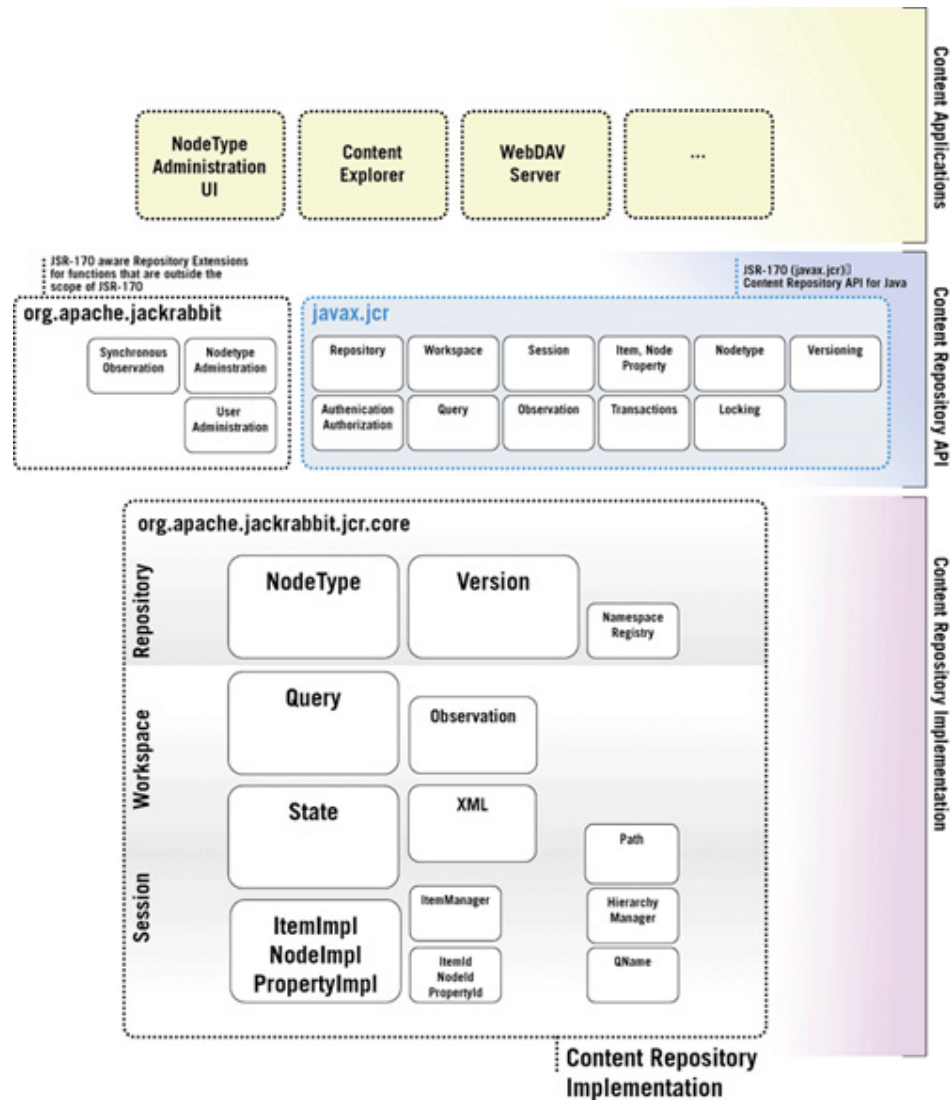
The logo graphic consists of a vertical black line intersecting a horizontal black line. To the left of the intersection, there are three overlapping squares: a yellow one at the top, a red one in the middle, and a blue one at the bottom. The word "Jackrabbit" is written in a blue, sans-serif font to the right of the graphic.

# Jackrabbit

---

- First version released in in April 2006 (1.0).
- Current release is 1.3.1, released July 31 2007.
- Version 2.0, which supports JSR 283 is targeted for 2008.

# Jackrabbit



The logo graphic consists of overlapping colored squares (yellow, red, blue) and a black crosshair.

# Jackrabbit

---

- Sample code :
  - Open a repository and log into it to get a Session.

```
Repository repository = new TransientRepository();  
session = repository.login(new SimpleCredentials("username", "password".toCharArray()));
```

- This will create a new repository and return a session.

A graphic consisting of overlapping colored squares (yellow, red, blue) and a black crosshair.

# Jackrabbit

---

- Inserting data
  - Create/Open a Repository.
  - Get a Session.
  - Get the Node that you want to add it to.
  - Create a Node.
    - `addNode(..)`
  - Set properties.
  - Save the Session.
  - Log out the session.

The logo graphic consists of a vertical black line intersecting a horizontal black line. To the left of the vertical line, there are three overlapping squares: a yellow one at the top, a red one in the middle, and a blue one at the bottom. The word "Jackrabbit" is written in a blue, sans-serif font to the right of the vertical line.

# Jackrabbit

---

- Note :

- Jackrabbit's dummy access manager (SimpleAccessManager) restricts the anonymous user to read-only access. Any other user is given full access.
- So in order to insert/modify/delete nodes you must log into the repository.



# Jackrabbit

---

- Example of insertion

```
...
// Get a repository
Repository repository = new TransientRepository();
// Get the session
Credentials credentials = new SimpleCredentials("hermod",
"css2007".toCharArray());
Session session = repository.login(credentials);
// Obtain the root node
Node rn = session.getRootNode();
// Create and new node
Node n = rn.addNode("Node1");
n.setProperty("NodeName",
new StringValue("First node"));
// Add the node by saving the session
session.save();
session.logout();
...
```

The logo graphic consists of a vertical black line intersecting a horizontal black line. To the left of the intersection, there are three overlapping squares: a yellow one at the top, a red one in the middle, and a blue one at the bottom. The word "Jackrabbit" is written in a blue, sans-serif font to the right of the vertical line.

# Jackrabbit

---

- Retrieving data
  - Open a Repository.
  - Get a Session.
  - Get the root node.
  - Get the node(s) from the rootnode.
    - getNode(..)
    - getNodes(..)
  - Log out the session.



# Jackrabbit

---

- Example of retrieval:

```
...
// Get a repository
Repository repository = new TransientRepository();
// Get the session
Credentials credentials = new SimpleCredentials("hermod", "css2007"
.toCharArray());
Session session = repository.login(credentials);
Node rn = session.getRootNode();
// Create and new node
Node n = rn.addNode("Node1");
n.setProperty("nodeName",
new StringValue("First node"));
// Add the node by saving the session
session.save();
// Get the node we inserted
Node n2 = rn.getNode("Node1");
System.out.println("Node name: " + n2.getProperty("nodeName").getString());
session.logout();
...
```

The logo graphic consists of a vertical black line intersecting a horizontal black line. To the left of the intersection, there are three overlapping squares: a yellow one at the top, a red one in the middle, and a blue one at the bottom. The word "Jackrabbit" is written in a blue, sans-serif font to the right of the graphic.

# Jackrabbit

---

- Removing data:
  - Open a Repository.
  - Get a Session.
  - Get the root node.
  - Get the node from the rootnode.
  - Remove the node.
    - remove()
  - Save the session.
  - Log out the session.



# Jackrabbit

---

- Example of removing node:

```
// Get a repository
Repository repository = new TransientRepository();
// Get the session
Credentials credentials = new SimpleCredentials("hermod", "css2007"
.toCharArray());
Session session = repository.login(credentials);
Node rn = session.getRootNode();
// Create and new node
Node n = rn.addNode("Node1");
n.setProperty("NodeName", new StringValue("First node"));
// Add the node by saving the session
session.save();
NodeIterator ni=rn.getNodes("Node1");
//Remove the node(s) we added
while(ni.hasNext())
{
n=ni.nextNode();
System.out.println("About to remove: " + n.getName());
n.remove();
}
session.save();
session.logout();
```

The logo graphic consists of overlapping colored squares (yellow, red, blue) and a black crosshair.

# Jackrabbit

---

- Versioning data
  - Open a Repository.
  - Get a Session.
  - Get the root node.
  - Create a node from the rootnode.
  - Add the nodetype `mix:versionable` to the node.
  - Add some properties.
  - Save the session.
  - Check in the node



# Jackrabbit

---

- Example of versioning data:

```
// Get a repository
Repository repository = new TransientRepository();
// Get the session
Credentials credentials = new SimpleCredentials("hermod", "css2007"
.toCharArray());
Session session = repository.login(credentials);
// Get root node
Node rn = session.getRootNode();
// Create versionable node
Node n = rn.addNode("Conference", "nt:unstructured");
n.addMixin("mix:versionable");
n.setProperty("Name", "Colorado Software Summit");
session.save();
n.checkin();
// Add new version to the node
Node versionedNode = rn.getNode("Conference");
versionedNode.checkout();
versionedNode.setProperty("Year", "2007");
session.save();
versionedNode.checkin();
```

The logo graphic consists of a vertical black line intersecting a horizontal black line. To the left of the intersection, there are three overlapping squares: a yellow one at the top, a red one in the middle, and a blue one at the bottom. The word "Jackrabbit" is written in a blue, sans-serif font to the right of the vertical line.

# Jackrabbit

---

- Some more samples..
  - Creating your own node types:
    - A Blog



# JSR 170

---

- Creating node types
  - Take care when modeling a hierarchical database.
  - Don't think "Relational".
  - Do you need to keep related information or do they only give meaning bound to the parent?



# JSR 170

---

- Creating node types
  - Compact Namespace and Node Type Definition (CND).
  - First the namespace:
    - `<ns = 'http://somenamespace.com/ns'>`
  - Then the name
    - `[ns:NodeType]`
  - Then optionally supertypes
    - `> ns:ParentType1, ns:ParentType2`



# JSR 170

---

- Then supported attributes:
  - ordeable
  - mixin
  
- Then define the properties of the node type:
  - Name of the property
  - Required type:
    - ✓ STRING
    - ✓ BINARY
    - ✓ LONG



# JSR 170

---

- ✓ DOUBLE
- ✓ DATE
- ✓ PATH
- ✓ NAME
- ✓ REFERENCE (references another node)
- ✓ UNDEFINED (can be of any type)
- Value constraints:
  - ✓ The range of values that may be assigned to this property.
- Default Value
  - ✓ Value that will automatically be set if no value is assigned.



# JSR 170

---

- Auto-create Status
  - ✓ Property will be auto-created when its parent node is created.
- Mandatory Status
  - ✓ Property must exist. Can not save node before it is set. Multi-value properties may have zero or more values.
- On-Parent-Version Status
  - ✓ Specifies what happens to this property if a new version of its parent node is created.
- Protected Status
  - ✓ Property cannot be modified.



# JSR 170

---

- Multiple Values Status

- ✓ Property can have multiple values – *i.e.* array of values.

➤ Then define any Child nodes.

- Name

- ✓ Name of the child node.

- Required Primary Types

- ✓ Specifies which nodes type(s) may be added.

- Default primary type

- ✓ Node type automatically assigned if no node type information is specified when the node is created.



# JSR 170

---

- Auto-create Status
  - ✓ If child node will be automatically added when parent is created. Mandatory Status
- Mandatory status
  - ✓ If the child node is mandatory.
- On-Parent-Version Status
  - ✓ What to do when the parent is versioned
- Protected Status
  - ✓ If the child node can be changed or not
- Same-Name Siblings Status
  - ✓ If more than one occurrence of the child node with same name may exist.



# JSR 170

---

- A blog node type



# JSR 283

---

- What is JSR 283?
  - Version 2.0 of Content Repository for Java.
  - Includes greater access control management, workspace and node-type management in addition to retention aspects of content or cross repository aspects.



# JSR 283

---

- Provides improved interoperability within the content repository through the addition of new standardized node types like meta information and internationalization.
- Improved XPath and XML query support.
  - Now support XPath 2.0.



# JSR 170

---

- **References:**

- <http://www.cmswiki.com/tiki-index.php?page=repository>
- <http://www.day.com/site/en/index.html>
- <http://jcp.org/en/jsr/detail?id=170>
- <http://jackrabbit.apache.org/>



# JSR 170

---

