JSR-170 1.0.1 Maintenance Release
Proposed Changes

Issue number and summary refer to the internal expert group issue tracker records.

#	Issue Summary	Proposed Resolution
20	Forward-Fit 1.0.1 Issue - NamespaceRegistry: Prefix Reassignment	6.3.3 Session Namespace Remapping Session.setNamespacePrefix javadoc Replace Within the scope of this Session, remaps a persistently registered namespace URI to the new prefix. The remapping only affects operations done through this session. To clear all remappings, the client must acquire a new Session. A prefix that is currently already mapped to some URI (either persistently in the repository NamespaceRegistry or transiently within this Session) cannot be remapped to a new URI using this method, since this would make any content stored using the old URI unreadable. An attempt to do this will throw a NamespaceException. As well, a NamespaceException will be thrown if an attempt is made to remap an existing namespace URI to a prefix beginning with the characters “xml” (in any combination of case). A NamespaceException will also be thrown if the specified uri is not among those registered in the NamespaceRegistry. with If existingUri is not registered in the NamespaceRegistry a NamespaceException will be thrown. If newPrefix is already locally mapped to existingUri (i.e., within this Session, by virtue of an earlier setNamespaceRegistry call) then this method returns silently and has no effect. If newPrefix is already locally mapped to a URI other than existingUri, then that URI reverts to its globally mapped prefix (as set in the NamespaceRegistry) and newPrefix is locally mapped to existingUri. If newPrefix is already assigned in the global NamespaceRegistry to otheruri (which differs from existingUri) and otherUri has not been locally mapped to another prefix which differs from newPrefix, then a NamespaceException will be thrown. In order to successfully locally map newPrefix to existingUri, otherUri must first be locally mapped to another prefix. 7.2 Adding and Deleting Namespaces NamespaceRegistry.registerNamespace javadoc bullet 2 Replace • Attempting to re-assign a prefix that is currently assigned to a URI that is present in content (either within an item name or within the value of a NAME or PATH property) will throw a

		NamespaceException. This includes prefixes in use within in-content node type definitions. with • Attempting to re-assign a prefix that is currently assigned to an “in-use” URI, i.e., one that is present in content, will throw a NamespaceException. This applies to URIs in use within item names and those within the values of NAME or PATH properties (including those in in-content node type definitions). However, one can change the prefix for an existing URI to any available new unique prefix, thus replacing the existing shorthand for that URI.
21	Forward-Fit 1.0.1 Issue 700 - Name/Path Grammar: Remove Duplication	6.2.5.3 Path Replace redundant name production with reference to name production already stated in 6.2.5.2 Name
23	Forward-Fit 1.0.1 Issue 697 - Escaping of Names Inconsistency (submitted by David Pitfield)	6.4.3 Escaping of Names Replace So, for example, • “My Documents” is converted to “My_x0020_Documents”, • "My_Documents” is not encoded, • “My_x0020Documents” is not encoded either, • but “My_x0020_Documents” is encoded as “My_x005f_x0020_Documents”. with So, for example, • “My Documents” is encoded as “My_x0020_Documents”. • “My_Documents” is not encoded. • “My_x0020Documents” is encoded as “My_x005f_x0020Documents”. • “My_x0020_Documents” is encoded as “My_x005f_x0020_Documents”. • “My_x0020 Documents” is encoded as “My_x005f_x0020_x0020_Documents”.
29	According to BNF "/" is not a valid path. It should be.	4.6 Path Syntax, 6.2.5.2 Name and 6.2.5.3 Path Fixed BNF
30	According to BNF paths terminating in "/" are valid. RI considers them invalid.	4.6 Path Syntax, 6.2.5.2 Name and 6.2.5.3 Path Fixed BNF
31	Clarify definition of "name"	4.6.1 Names vs. Paths Replace A “name” in this specification is a path element without any square-bracket index. with A “name” is valid if satisfies the above name production. It can be thought of, informally, as a single path element without any square-bracket index (and not including the '.' and '..').

32	Correct Name BNF wrt ".." and "."	4.6 Path Syntax and 6.2.5.2 Name Fixed BNF
39	Clarify why "required node types" (plural) makes sense	6.7.4 Primary and Mixin Node Types Replace In a content repository, every node has one and only one primary node type. This node type defines, as mentioned, a set of restrictions on the child items of the node. with In a content repository, every node has one and only one declared primary node type. This node type defines, as mentioned, a set of restrictions on the child items of the node. Note that because node types may inherit characteristics from supertypes, a particular node may be of more than one primary node type by virtue of type inheritance. For example if X is a supertype of Y and node N is of type Y, then N is also of type X. Nonetheless, this does not change the fact that any particular node still has exactly one declared node type. 6.7.7 Child Node Definitions bullet 2 Added A child node definition may, for example, specify required node types X and Y. This does not mean that the node specified will have more than one declared primary node type, but rather that its primary node type (whatever else it may be) must be at least a subtype of both node types X and Y. Of course, inheritance is also respected in the simpler case where this attribute specifies only one primary node type. If it specifies, for example, type T, this means that the child node must be of type T or a subtype of type T. Finally, it should be clear that the subtype relationship between the required type (or types) and the actual type of the child node must be an explicit one, that is, it must arise be by virtue of a chain of declared superclass attributes
40	Clarify semantics of node type inheritance	6.7.8 Inheritance Among Node Types Replace A node type may have one (or in some implementations, more than one) supertype. A subtype inherits the property and child node definitions of its supertype(s) (and possibly other attributes) and may declare further property or child node definitions. with The semantics of inheritance follow the usual rules: • The supertype relation is, as one would expect, transitive. In other words if T1 is a supertype of T2 and T2 is a supertype of T3 then T1 is a supertype of T3. • The subtype relation is, of course, the converse of supertype: T1 is

		a subtype of T2 if and only if T2 is a supertype of T1. Hence, subtype is also a transitive relation. • The ‘is of type’ relation which holds between node instances and node types (as in, node N is of type T) is itself transitive across the subtype relation. In other words, if T2 is a subtype of T1 and N is of type T2 then N is also of type T1. This predicate appears in the API as the method Node.isNodeType. Note that this relation is also the one that is relevant in the child node definition attribute required primary node types. • The supertype relation always and only stems from explicit Supertypes attribute declarations within the set of node types. For example, just because the property and child node definitions of T2 happen to be a superset of those of T1 does not make T1 a supertype of T2. For that to be the case, T2 must declare T1 as its supertype. • Similarly, the ‘is of type’ relation always and only stems from an explicit assignment of a node type to a node. Just because node N happens to have the properties and child nodes declared by node type T does not necessarily mean that N is of type T. For that to be the case, N must have been explicitly assigned the type T, or a subtype of T. Management of the hierarchy of node types available within a particular repository is outside the scope of this specification. However, the requirement of preserving the ‘is of type’ relation across subtyping, as mentioned above, does imply certain things about inheritance. The requirement can be restated as: • If T2 is a subtype of T1, then any instance of T2 must also be a valid instance of T1. Note that an implementation can meet this requirement in a number of ways, ranging from the most coarse-grained to the most fine-grained. A coarse-grained approach would be to say that a subtype can never override the property of child node definition of a supertype (that is, declare a definition with the same name as one in the supertype). A more fine-grained approach would allow such overrides, but only in cases where an instance of the subtype would still be a valid instance of the supertype. For example, if a supertype declares a property definition called A of type UNDEFINED, a subtype would may override that with a property definition A of type STRING. However, the reverse would not be allowed. For purposes of the above, the notion of two definitions having the same name does not apply to two residual definitions. Two (or more) residual property or child node definitions with differing sub-attributes must be permitted to co-exist in the same effective node type. They are interpreted as disjunctive (ORed) options. Apart from the issue of how inheritance affects the set of property and child node definitions, there is also the issue of the top-level node type attributes mixin-status, orderable-status and primary item. The specification implies only one requirement with regard to these

		attributes: that a mixin node-type is capable of being a supertype of a primary node type, and therefore that a mixin-status of primary in the subtype overrides a mixin-status of mixin in the supertype (See, 6.7.22.2 Additions to the Hierarchy). Other than this, the specification does not define how conflicts between multiple supertypes are resolved or how these three top level attributes are inherited. For example, the question of whether the orderable child nodes setting of a node type is inherited by its subtypes is left up to the particular implementation.
44	Inaccurate to say that direct methods "do a save". They do not.	Added new section 7.1.3.7 Save vs Direct The direct-to-workspace methods should not be thought of automatically “doing a save”. The effect of a directly-persistent method happens one level deeper and is therefore not necessarily equivalent to making the same change transiently and then immediately calling save. For example: Following A.addNode("B") we have: A (transient state differs from persistent because of B) | |--B (transient) If we now import C as child of A (using the importXML method, though this would apply to any direct-to-workspace child addition) we have: A (transient state still differs from persistent because | of B) | |--B (transient) | |--C (persistent) If we now do an A.save we get an InvalidItemStateException because A's state on the persistent layer has changed due to the import. We would now have to do an A.refresh(false), discarding the effect of the A.addNode("B") and reverting the situation to: A (persistent) | |--C (persistent) At this point we could do the A.addNode("B") again and, as long as we saved it before making an further direct-to-workspace changes to A, the save would succeed. 8.2.14.1 Node Versioning Methods Node.checkin javadoc Replace If checkin succeeds, the change to the jcr:checkedOut property is automatically saved (there is no need to do an additional save).

		with If checkin succeeds, the change to the jcr:checkedOut property is automatically persisted (there is no need to do an additional save).
45	Fix versioning explanation	8.2.4 Initializing the Version History bullet 4 Replace • V0 is the root version of VH. This root version does not contain any state information about N other than the node type and UUID information in the jcr:frozenPrimaryType, jcr:frozenMixinTypes, and jcr:frozenUuid. It is a dummy version. with • V0 is the root version of VH. It is a dummy version that serves as the starting point of the version graph. Like all version nodes, it has a subnode called jcr:frozenNode. But, in this case that frozen node does not contain any state information about N (other than the node type and UUID information found in the properties jcr:frozenPrimaryType, jcr:frozenMixinTypes, and jcr:frozenUuid).
46	Typo	6.2.3 Read Methods Replace reference to 7.1.2.1 Re-using Item Objects. with reference to 7.1.3.1 Re-using Item Objects.
47	Inaccurate definition of 'prefix'	4.6 Path Syntax and 6.2.5.2 Name BNF Replace XML Name with XML NCName
49	getPrimaryItem can be ambiguous in cases where the specified item is a node with same-name-siblings.	6.2.3 Read Methods Node.getPrimaryItem javadoc Added In cases where the primary child item specifies the name of a set of same-name sibling child nodes, the node returned will be the one among the same-name siblings with index [1].
50	Lock should not apply to new unsaved node	8.4.11 Locking Methods Node.lock javadoc Added If this node does not have a persistent state (has never been saved), a LockException is thrown.
54	What happens when a registered namespace is "hidden" by a later	6.3.3.3 Internal Storage of Names and Paths Section renamed to

	registration	6.3.3.3 Conflict between Session Remapping and Namespace Registry and the following added before the existing text in that section There are two circumstances in which a potential conflict may arise between a session namespace mapping and the repository-wide namespace registry. The first case occurs when an attempt is made to locally map prefix P to URI U where P is already globally mapped to URI U' (not equal to U) and there is no local mapping of some prefix P' (not equal to P) to U'. As described above, an exception is thrown in this case. To successfully locally map P to U, U' must first be locally mapped to some P'. The second case occurs when a mapping of P to U is added to the global namespace registry while P is locally mapped to some U' (not equal to U) in at least one active session S. How this conflict is handled is left up to the implementation, since in any case, the mechanism for making changes to the global namespace registry is beyond the scope of this specification.
56	Clarify that NS changes are not within transaction	Added new section 6.3.4 Transactions and Namespaces In repositories that support transactions, both changes to the namespace registry and Session namespace remappings must be non-transactional.
57	'IN' SQL operand, does it exist or not? [Answer: No]	8.5.3 SQL EBNF Fixed BNF
58	Throws clause of Session.itemExists should be aligned with that of Session.getItem	6.2.1 Session Read Methods Session.itemExists javadoc Replace Returns true if an item exists at absPath and this Session has read access to it; otherwise returns false. Also returns false if the specified absPath is malformed. A RepositoryException is thrown if an error occurs. with Returns true if an item exists at absPath and this Session has read access to it; otherwise returns false. A RepositoryException is thrown if absPath is not a well-formed absolute path.
60	Remapping the default namespace and prefix	6.3.3 Session Namespace Remapping Replace Any registered namespace can be temporarily remapped to a new prefix within the scope of a particular Session. with

		Any registered namespace (other than the empty namespace or one beginning with "xml") can be temporarily remapped to a new prefix within the scope of a particular Session.
61	Invalid characters in string properties	6.4.1 System View XML Mapping point 7 Added In addition, if the string form of a value contains characters which cannot appear in XML documents at all (neither as literals nor as character references) then the value is also Base64 encoded, the attribute xsi:type="xsd:base64Binary" is added to the <sv:value> element, and the namespace mappings for xsi and xsd are added to the topmost XML element (i.e., xmlns:xsd="http://www.w3.org/2001/XMLSchema" and xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"). Note that since BINARY values are always Base64 encoded, the xsi:type="xsd:base64Binary" attribute is considered the default in those cases and therefore is omitted. and added new section 6.4.2.6 Invalid Characters in Values If the string form of the value of property P contains characters which cannot appear in XML documents at all (neither as literals nor as character references) then the attribute P is simply excluded from the document view serialization and does not appear at all.
63	Clarify that a child node and property of a given node cannot have the same name.	4 The Repository Model Replace Any item in the hierarchy can be identified by an absolute path. with Every item in a workspace apart from the root node has a non-empty name. The root node always has the empty string as its name. A property and node with the same parent cannot have the same name. Two or more nodes with the same parent may in some circumstances have the same name, in which case they are distinguished by index (see 4.3 Same-Name Siblings for more details). The chain of names from the root to any item defines the absolute path of that item.
66	Clarify how to set NAME and PATH properties	7.1.5 Adding and Writing Properties setProperty javadoc (the last entry; with multiple signatures listed) Replace To create a property of PropertyType.NAME or PropertyType.PATH an explicit type must be specified using a three-argument signature. with A property of type PropertyType.NAME or PropertyType.PATH may be created either by explicitly specifying the property type using a

		three-argument setProperty signature, or by using ValueFactory to create a property of the desired type and then calling setProperty(String, Value).
73	Item.isSame: clarify what constitutes "sameness"	6.2.8 Item Read Methods Item.isSame javadoc Added Two Item objects represent the same repository item if all the following are true: • Both objects were acquired through Session objects that were created by the same Repository object. • Both objects were acquired through Session objects bound to the same repository workspace. • The objects are either both Node objects or both Property objects. • If they are Node objects, they have the same correspondence identifier (see 4.10.2 Multiple Workspaces and Corresponding Nodes). • If they are Property objects they have identical names and their parent nodes have the same correspondence identifiers.
75	jcr:mergeFailed should be constrained to be nt:version	6.7.21.3 mix:versionable and 8.2.1 Versionable Nodes mix:versionable node type definition Added value constraint
77	Move or copy semantics of "." and ".." from 4 to 4.6	Added new section 4.6.2 Current Item and Parent Item The syntax of paths includes the segments “.” and “..” indicating current item and parent item, respectively. These can be used within JCR paths just as they can in Unix-like file system paths. For example, /a/b/../c is equivalent to /a/c while /a/b/./c is equivalent to /a/b/c.
78	Various typos in the spec	6.7.21.3 mix:versionable, 8.2.1 Versionable Nodes, 8.2.2.3 nt:versionHistory and 8.2.10.1 Merge Algorithm Fixed typos
82	Wildcard in XPath syntax should be required	6.6.6.2 Non-Terminals production 56 Removed strikethrough
83	Session.removeLockToken() should fail on a session-scoped lock	8.4.7 Session-scoped and Open-scoped Locks Added Additionally, since a session-scoped lock is always tied to the session that created it, it does not make sense to move the token of session-scoped lock from its originating session to some other session. Consequently, Session.removeLockToken(token) will

		always fail when token specifies a session-scoped lock (see 8.4.13 Session Methods Related to the Lock Token). 8.4.13 Session Methods Related to the Lock Token Session.removeLockToken javadoc Added A LockException is thrown if the lock associated with the specified lock token is session-scoped. A RepositoryException is thrown if another error occurs.
84	Session.addLockToken should fail if that lock token is currently held by another session	8.4.13 Session Methods Related to the Lock Token Session.addLockToken javadoc Added A LockException is thrown if the specified lock token is already held by another Session. A RepositoryException is thrown if another error occurs.
85	Errors in SQL examples	6.6.3.5 Ordering Specifier Examples table Replace SELECT * FROM my:type WHERE CONTAINS(., 'jcr') with SELECT * FROM my:type WHERE CONTAINS(*, 'jcr') 8.2.2.2 Searching and Traversing Version Storage Replace SELECT * FROM nt:version WHERE productName = "Car" AND price > "30000" AND jcr:path LIKE "/jcr:system/jcr:versionStorage/%" with SELECT * FROM nt:version WHERE productName = 'Car' AND price > 30000 AND jcr:path LIKE '/jcr:system/jcr:versionStorage/%'
86	Missing terminal for jcr:path join	8.5.3 SQL EBNF Fixed BNF
87	Typo: comments delimiters in BNF inconsistent	4.6 Path Syntax and 6.2.5.2 Name Fixed BNF
88	Binary values to Base64 in Doc View	6.4.2.1 General Structure point 6

		Replace 6. The value of each property P is converted to string form according to the standard conversion (see 6.2.6 Property Type Conversion) and becomes the value of the XML attribute P. Entity references are used to escape characters which should not be included as literals within attribute values (see 6.4.4 Escaping of Values). with 6. If P is a non-BINARY property its value is converted to string form according to the standard conversion (see 6.2.6 Property Type Conversion). If P is a BINARY property its value is Base64 encoded. The resulting string becomes the value of the XML attribute P. Entity references are used to escape characters which should not be included as literals within attribute values (see 6.4.4 Escaping of Values).
91	Version number and Date are wrong	0 Title page, 1 Preface Version number incremented to 1.0.1, date changed, mention made in Preface of status as Maintenance Release.
92	Explanation of transient storage fails to mention basics	7.1.1.1 Writing to Transient Storage Added To persist any change that involves the addition, removal or re-ordering of nodes or the addition or removal of properties, the scope of the save must include the parent node affected. To persist a change that involves only a change to the value of an existing property, only that property itself needs to be within the scope of the save. Note that this means that the minimal scope required to persist the effect of Node.setProperty depends on whether the property in question already exists or not. If it does not, then the parent node must be saved. If it does, then only the property itself needs to be saved.