Concept - metaData content & nodeTypeDefinition migration task

- Introduction
    - Migrate JCR NodeTypeDefinition.
    - Migrate JCR content
    - Identify and rename siblings node name
- Tasks
    - Migrate NodeTypeDefinition.
    - Remove metaData sub nodes in nodeType definition
    - Remove metaData sub nodes in content repository
    - Identify and rename same name sibling nodes

Introduction

In Magnolia M5, the metaData subnode is gone (reason and initial task) and replaced by newly introduce MixIn NodeTypeDefinition. In addition, M5.2 nodeType definition do not allows anymore sibling names (two or more nodes having the name name and parent)
Thus we have three main migration task:

- Migrate JCR NodeTypeDefinition
- Migrate JCR content with metaData nodes (replace these nodes by mixIn's)
- Identify and rename siblings node name.

Migrate JCR NodeTypeDefinition.

For a fresh installation of M5, the NodeTypeDefinition are defined into the following xml file (`/src/main/resources/mgnl-nodetypes/magnolia-nodetypes.xml`).
Unfortunately, when uprating from M4.5.x to M5.x, NodeTypeDefinition are already defined based on the M4.5.x `magnolia-nodetypes.xml`. and we need to run a Task that adapt the NodeTypeDefinition.

M5 core module define a version handler that for update to version 5.0, automatically update NodeTypeDefinition.

This update will:

- Create the new mixIn NodeTypeDefinition
- Update existing NodeTypeDefinition
    - Add mixIn as super (parent)
    - Remove child node (metaData)

Migrate JCR content

M5 core module define a version handler that for update to version 5.0, automatically update all visible workspace.

This update will:

- Converts the metaData sub node into properties on the mixins
    - mgnl:created,
    - mgnl:lastModified
    - mgnl:renderable
    - mgnl:activatable
    - mgnl:activatable
- Renames the property mgnl:deletedOn on the mixin mgnl:deleted
- The metaData node itself is optionally removed if there are no additional properties on it.

In addition this update is triggered during Import process. Meaning that every xml files imported into a repository is scanned and filter out before being imported.

Identify and rename siblings node name

M5.2 nodeTypde definition do not allow same name sibling.

node type definition

```
 <nodeType name="mgnl:content" isMixin="false" hasOrderableChildNodes="true" primaryItemName="">
    <supertypes>
      <supertype>nt:hierarchyNode</supertype>
      <supertype>mix:referenceable</supertype>
          ...
    </supertypes>
    <childNodeDefinition name="*" defaultPrimaryType="" autoCreated="false" mandatory="false" onParentVersion="
COPY" protected="false" sameNameSiblings="false">
    <!-- UNTIL M5.2 -->
    <!--
    <childNodeDefinition name="*" defaultPrimaryType="" autoCreated="false" mandatory="false" onParentVersion="
COPY" protected="false" sameNameSiblings="true">
    -->
    ...
  </nodeType>
```

Thus we provide two migration tasks:

1. Identify all sibling for specific node types, workspace, and path.
2. Rename all sibling for specific node types, workflow, and path.

Tasks

Migrate NodeTypeDefinition.

GOAL: Update the existing nodeTypeDefinition during upgrade from M4.5.x to M5.x.

⚠️ The new mixIn nodeType definition introduced by M5 do not have to be specifically registered. New node type are automatically registered.
Only modified nodeType definition have to be explicitly re-registered.

REQUIREMENTS:

- Task has to be reusable for custom NodeType registration
- Update the existing NodeTypeDefinition
- Create the new M5.x mixIn NodeTypeDefinition
- Unregister any desired NodeTypeDefinition

STEPS:

1. register NodeTypeDefinition.
    a. Iterate the NodeTypeDefinition list to register
        i. nodeType is already registered
            1. unregister the existing nodeType (and all his child nodeType)
            2. register the new nodeType (and all his child nodeType)
            3. **in case of exception**
                a. the previous nodeType is keeped
                b. a log message is displayed into the update screen
        ii. nodeType is not yet registered
            1. simply register the new nodeType
    b. Iterate the NodeTypeDefinition list to unregister
        i. Unregister the complete list
        ii. **in case of exception**
            1. Not a single nodeType of the list are unregistered.
            2. a log message is displayed into the update screen

Class: `AbstractNodeTypeRegistrationTask`
Constructor attributes:

Property	Description	Default value	Valid values
`taskName`	Task name used by the reporting tool, and to log/display informations related to this task.		String
`taskDescription`	Task Description used to display informations in the admin. central update view.		String

`workspaceName`	Name of the workspace from where the `NodeTypeManager` is retrieve		String

Abstract methods:

Methods	Description
`public abstract LinkedList<NodeTypeDefinition> getNodeTypeToRegister(NodeTypeManager nodeTypeManager)`	Define the list of `NodeTypeDefinition`. to register. Use `NodeTypeUtil` to create these definitions.
`public abstract LinkedList<String> getNodeTypeToUnRegister (NodeTypeManager nodeTypeManager)`	Define the list of `NodeTypeDefinition names` to unregister.

Implemented class

`Register50NodeTypeTask` define the list of `NodeTypeDefinition`. to register.

Custom implementation

Create your own NodeTypeMigration task

If you have created custom NodeTypeDefinition, you may need to migrate them.

```
/**
 * Register custom nodeType definition.
 */
public class RegisterCustomNodeTypeTask extends AbstractNodeTypeRegistrationTask {

        @Override
    public LinkedList<NodeTypeDefinition> getNodeTypeToRegister(NodeTypeManager nodeTypeManager) {
        LinkedList<NodeTypeDefinition> res = new LinkedList<NodeTypeDefinition>();

                // Define the nodeTypeDefinition
                NodeTypeTemplate nodeType = NodeTypeUtil.createNodeType(nodeTypeManager,  "custom:
baseNode", new String[] { "nt:base" }, false, true, null, true);
                // Add a child definition
        NodeDefinitionTemplate child = createChildNodeDefinition(nodeTypeManager, false, false, false,
true, null, null, OnParentVersionAction.COPY, new String[] { "nt:base" });
        nodeType.getNodeDefinitionTemplates().add(child);
                 // Add propertydefinition
        PropertyDefinitionTemplate propertyDefinitionNotMultiple = createPropertyDefinition(...
                nodeType.getPropertyDefinitionTemplates().add(propertyDefinitionNotMultiple);
                ...
                res.add(nodeType);
                ...
        }

    @Override
    public LinkedList<String> getNodeTypeToUnRegister(NodeTypeManager nodeTypeManager) {
        LinkedList<String> res = new LinkedList<String>();
        // Add NodeTypeDefinition name
        res.add("custom:old");
        return res;
    }
```

Remove metaData sub nodes in nodeType definition

GOAL: Clean the existing nodeType definition in order to remove metaData defined as `childNodeDefinition`.

REQUIREMENTS:

- none

STEPS:

1. Iterate all registered nodeType.
    a. For every nodeType that defines metaData as `childNodeDefinition`:
        i. Remove this `childNodeDefinition`

Class: `RemoveMetaDataInNodeTypeDefinitionTask`
Constructor attributes:

Property	Description	Default value	Valid values
`taskName`	Task name used by the reporting tool, and to log/display informations related to this task.		String
`taskDescription`	Task Description used to display informations in the admin. central update view.		String
`workspaceName`	Name of the workspace from where the `NodeTypeManager` is retrieve		String

Remove metaData sub nodes in content repository

GOAL: Clean the existing workspace during upgrade from M4.5.x to M5.x.

REQUIREMENTS:

- none

STEPS:

1. Iterate all registered workspace.
    a. For every workspace the task is traversing all nodes and filter, meaning:
        i. Converts the metaData sub node into properties on the mixins
        ii. Renames the property mgnl:deletedOn on the mixin mgnl:deleted
        iii. Remove the metaData node itself if there are no additional properties on it.

Class: `ConvertMetaDataUpdateTask`
Constructor attributes:

Property	Description	Default value	Valid values
`taskName`	Task name used by the reporting tool, and to log/display informations related to this task.		String
`taskDescription`	Task Description used to display informations in the admin. central update view.		String

Identify and rename same name sibling nodes

GOAL: Identify and or rename same name siblings node.

REQUIREMENTS:

- Perform the operation for a defined:
    - workspace
    - nodeType
    - sub path
- Let the naming convention used to rename sibling nodes be configurable.

STEPS:

1. Using the visitor pattern, based on the workspace, sub folder and for every node types
   Iterate all selected nodes (sibling).
    a. Log in the install screen all siblings node path and name
    b. In case of rename, simply rename the sibling name and Log the new name set.

Class: `IdentifySameNameSiblingNodesTask`
Constructor attributes:

Property	Description	Default value	Valid values
`taskName`	Task name used by the reporting tool, and to log/display informations related to this task.		`String`

`taskDescription`	Task Description used to display informations in the admin. central update view.		`String`
`workspace`	Name of the workspace from where the sibling nodes are search.		`String`
`subpath`	Name of the sub-path from where the sibling nodes are search.	`/`	`String`
`nodeTypes`	List name of the nodeTypes for witch the sibling nodes are search.	`nt:base`	`List<String>`
`evaluateSupertypes`	If set to true, the child nodeTypes of a defined 'nodeType' are also part of the evaluation.		`boolean`

Class: `RenameSameNameSiblingNodesTask`
Constructor attributes:

Property	Description	Default value	Valid values
`taskName`	Task name used by the reporting tool, and to log/display informations related to this task.		`String`
`taskDescription`	Task Description used to display informations in the admin. central update view.		`String`
`workspace`	Name of the workspace from where the sibling nodes are search.		`String`
`subpath`	Name of the sub-path from where the sibling nodes are search.	`/`	`String`
`nodeTypes`	List name of the nodeTypes for witch the sibling nodes are search.	`nt:base`	`List<String>`
`evaluateSupertypes`	If set to true, the child nodeTypes of a defined 'nodeType' are also part of the evaluation.		`boolean`

In case a sibling node is found, this node will be renamed using the following pattern:

- original node name : myNode
- renamed node name (first sibling) : myNode_sibling_0
- renamed node name (second sibling) : myNode_sibling_1

If you with to use your own renaming pattern, simply extends `RenameSameNameSiblingNodesTask` like in the following example:

```
public class CustomRenameSameNameSiblingNodesTask extends RenameSameNameSiblingNodesTask {
    public CustomRenameSameNameSiblingNodesTask(String name, String description, String workspace, String
subPath, List<String> nodeTypes, boolean evaluateSupertypes) {
        super(name, description, workspace, subPath, nodeTypes, evaluateSupertypes);
    }
    @Override
    protected String createNewName(Node node) throws RepositoryException {
        // Custom pattern
        return node.getName() + "_" + RandomStringUtils.randomAlphabetic(12);
    }
}
```