Content Loading

Content Loading and Nodetype Support

Apache Sling provides support for initial content loading into a repository and for registering node types. The `sling-jcr-contentloader` bundle provides loading of content from a bundle into the repository and the `sling-jcr-base` bundle provides node type registration.

Initial Content Loading

Bundles can provide initial content, which is loaded into the repository when the bundle has entered the *started* state. Such content is expected to be contained in the bundles accessible through the Bundle entry API methods. Content to be loaded is declared in the `Sling-Initial-Content` bundle manifest header. This header takes a comma-separated list of bundle entry paths. Each entry and all its child entries are accessed and entered into starting with the child entries of the listed entries.

Adding this content preserves the paths of the entries as show in this table, which assumes a `Sling-Initial-Content` header entry of `SLING-INF/content`:

Entry	Repository Path
`SLING-INF/content/home`	`/home`
`SLING-INF/content/content/playground/en/home`	`/content/playground/en/home`

Bundle entries are installed as follows:

Entry Type	Installation method
Directory	Created as a node of type `nt:folder` unless a content definition file of the same name exists in the same directory as the directory to be installed. Example: A directory `SLING-INF/content/dir` is installed as node `/dir` of type `nt:folder` unless a `SLING-INF/content/dir.xml` or `SLING-INF/content/dir.json` file exists which defines the content for the `/dir` node.
File	Unless the file is a content definition file (see below) an `nt:file` node is created for the file and an `nt:resource` node is created as its `jcr:content` child node to take the contents of the bundle file. The properties of the `nt:resource` node are set from file information as available. If the file is a content definition file, the content is created as defined in the file. See below for the content definition file specification.

It is possible to modify the intial content loading default behaviour by using certain optional directives. Directives should be specified separated by semicolon. They are defined as follows:

Directive	Definition	Default value	Description
`overwrite`	`overwrite:=(true|false)`	`false`	The overwrite directive specifies if content should be overwritten or just initially added.
`overwriteProperties`	`overwriteProperties:=(true|false)`	`false`	The overwriteProperties directive specifies if content properties should be overwritten or just initially added.
`uninstall`	`uninstall:=(true|false)`	`overwrite`	The uninstall directive specifies if content should be uninstalled when bundle is unregistered. This value defaults to the value of the `overwrite` directive.
`path`	`path:=`*`/target/location`	`/`	The path directive specifies the target node where initial content will be loaded. If the path does not exist yet in the repository, it is created by the content loader. The intermediate nodes are of type `nt:folder`.
`checkin`	`checkin:=(true|false)`	`false`	The checkin directive specifies whether versionable nodes should be checked in.
`ignoreImportProviders`	`ignoreImportProviders:=list of extensions`	`empty`	This directive can be used to not run one of the configured extractors (see below).

Examples of these directives uses could be (assumes a Sling-Initial-Content header entry of SLING-INF/content):

Entry	Behaviour
`SLING-INF/content/home;overwrite:=true;uninstall:=true`	Overwrites already existing content in */home* and uninstalls the content when the bundle is unregistered.
`SLING-INF/content/home;path:=/sites/sling_website`	if */sites/sling_website* exists it loads the content into it. Otherwise, it loads the content into root node */*.
`SLING-INF/content/home;checkin:=true`	After content loading, versionable nodes are checked in.

Loading initial content from bundles

Repository items to be loaded into the repository, when the bundle is first installed, may be defined in four ways:

1. Directories
2. Files
3. XML descriptor files

4. JSON descriptor files

Depending on the bundle entry found in the location indicated by the Sling-Initial-Content bundle manifest header, nodes are created (and/or updated) as follows:

Directories

Unless a node with the name of the directory already exists or has been defined in an XML or JSON descriptor file (see below) a directory is created as a node with the primary node type "nt:folder" in the repository.

Files

Unless a node with the name of the file already exists or has been defined in an XML or JSON descriptor file (see below) a file is created as two nodes in the repository. The node bearing the name of the file itself is created with the
primary node type "nt:file". Underneath this file node, a resource node with the primary node type "nt:resource" is created, which is set to the contents of the file.

The MIME type is derived from the file name extension by first trying to resolve it from the Bundle entry URL. If this does not resolve to a MIME type, the Sling MIME type resolution service is used to try to find a mime type. If all fals, the MIME type is defaulted to "application/octet-stream".

XML Descriptor Files

Nodes, Properties and in fact complete subtrees may be described in XML files using either the JCR SystemView format, or the format described below. In either case, the file must have the .xml extension.

```
<node>
                <!--
                        optional on top level, defaults to XML file name without .xml extension
                       required for child nodes
                -->
                <name>xyz</name>

                <!--
                           optional, defaults to nt:unstructured
                -->
                <primaryNodeType>nt:file</primaryNodeType>

                <!--
                    optional mixin node type
                    may be repeated for multiple mixin node types
                -->
                <mixinNodeType>mix:versionable</mixinNodeType>
                <mixinNodeType>mix:lockable</mixinNodeType>

                <!--
                        Optional properties for the node. Each <property> element defines
                        a single property of the node. The element may be repeated.
                -->
                <property>
                        <!--
                                required property name
                        -->
                        <name>prop</name>

                        <!--
                                value of the property.
                                For multi-value properties, the values are defined by multiple
                                <value> elements nested inside a <values> element instead of a
                                single <value> element
                        -->
                        <value>property value as string</value>

                        <!--
                                Optional type of the property value, defaults to String.
                                This must be one of the property type strings defined in the
                                JCR PropertyType interface.
                        -->
                        <type>String</type>
                </property>

                <!--
                        Additional child nodes. May be further nested.
                -->
                <node>
                ....
                </node>
        </node>
```

If you're including binary files (nt:file) in your content, you may store them anywhere, and reference them from your XML by using nt:file elements. Use the <nt:file> instead of a <node> where you want your file to appear in your repository content:

```
  <nt:file src="url/relative/to/this/xml/image.jpg" mimeType="image/jpeg" lastModified="2009-10-27'T'10:50:
15+0100" />
```

You may leave out the lastModified attribute - if so, the file's last modified date as reported by the filesystem will be used.

Using a custom XML format

By writing an XSLT stylesheet file, you can use whatever XML format you prefer. The XML file references an XSLT stylesheet by using the xml-stylesheet processing instruction:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="my-transform.xsl" type="text/xsl"?> <!-- The path to my-transform.xsl is relative to
this file -->

<your_custom_root_node>
   <your_custom_element>
   ...
   </your_custom_element>
...
</your_custom_root_node>
```

The my-transform.xsl file is then responsible for translating your format into one of the supported XML formats:

```
<xsl:stylesheet version="1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:mix="http://www.jcp.org/jcr/mix/1.0"
  xmlns:sv="http://www.jcp.org/jcr/sv/1.0" xmlns:sling="http://sling.apache.org/jcr/sling/1.0"
  xmlns:rep="internal" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

  <xsl:template match="your_custom_element">
    <node>
      ...
    </node>
  </xsl:template>
  ...
</xsl:stylesheet>
```

JSON Descriptor Files

Nodes, Properties and in fact complete subtrees may be described in JSON files using the following skeleton structure (see http://www.json.org or information on the syntax of JSON) :

```
{
                // optional node name on top level, default is file name without .json ext.
                "name": "nodename",

                // optional primary node type, default "nt:unstructured"
                "primaryNodeType": "sling:ScriptedComponent",

                // optional mixin node types as array
                "mixinNodeTypes": [ ],

            // the "properties" property is an object indexed by property name whose
            // value is either the string property value, array for multi-values or
            // an object whose value[s] property denotes the property value(s) and
            // whose type property denotes the property type
            "properties": {
                    "sling:contentClass": "com.day.sling.jcr.test.Test",
                    "sampleMulti": [ "v1", "v2" ],
                    "sampleStruct": {
                            "value": 1,
                            "type": "Long"
                    }
                    "sampleStructMulti": {
                            "value": [ 1, 2, 3 ],
                            "type": "Long"
                    }
            },

            // the "nodes" property is an array of objects denoting child nodes. Nodes
            // may be further nested.
                "nodes": [
                        {
                                // the name property is required on (nested) child nodes
                    "name": "sling:scripts",

                                "primaryNodeType": "sling:ScriptList",

                                "nodes": [
                                        {
                                                "primaryNodeType": "sling:Script",
                                                "properties": {
                                                        "sling:name": "/test/content/jsp/start.jsp",
                                                        "sling:type": "jsp",
                                                        "sling:glob": "*"
                                                }
                                        }
                                ]
                        }
                ]
        }
```

Extractors

By default, the `sling-jcr-contentloader` bundle tries to extract certain file types during content loading. These include `json`, `xml`, `zip`, and `jar` files. Therefore all available extractors are used for content processing. However if some files should be put into the repository unextracted, the `ignoreImportProviders` directive can be used with a comma separated list of extensions that should not be extracted, like `ignoreImportProviders:=jar,zip`.

Workspace Targetting

By default, initial content will be loaded into the default workspace. To override this, add a `Sling-Initial-Content-Workspace` bundle manifest header to specify the manifest. Note that **all** content from a bundle will be loaded into the same workspace.

Declared Node Type Registration

The `sling-jcr-base` bundle provides low-level repository operations which are at the heart of the functionality of Sling:

- **Node Type Definitions** - The class `org.apache.sling.content.jcr.base.NodeTypeLoader` provides methods to register custom node types with a repository given a repository session and a node type definition file in CND format. This class is also used by this bundle to register node types on behalf of other bundles.

Bundles may list node type definition files in CND format in the `Sling-Nodetypes` bundle header. This header is a comma-separated list of resources in the respective bundle. Each resource is taken and fed to the `NodeTypeLoader` to define the node types.

After a bundle has entered the *resolved* state, the node types listed in the `Sling-Nodetypes` bundle header are registered with the repository.

Node types installed by this mechanism will never be removed again by the `sling-jcr-base` bundle. Likewise, registered node types cannot currently be modified using this feature. The `NodeTypeLoader` will try to load nodes defined and fail with a log message if a node type has already been defined. To update existing node type definitions, native repository functionality has to be used.

Nodetype management is currently a problematic issue, as the only API available is contained in the Jackrabbit Core library, which is generally not available to client applications - unless running in the same VM and class loader hierarchy as the Repository. Version 2 of the JCR Specification currently being developped as JSR-283 should fix this issue by providing an official node type management API. Until then, this approach is about the only solution we have.

Automated tests

The initial content found in the sling-test folder of the launchpad initial content is verified by the InitialContentTest when running the *launchpad/testing* integration tests.

Those tests can be used as verified examples of initial content loading. Contributions are welcome to improve the coverage of those tests.