XML Documentation for Adobe Experience Manager API Reference

December 10, 2019

Contents

Introduction

XML Documentation for Adobe Experience Manager (referred to as *XML Documentation solution* later in this guide) is an end-to-end, enterprise solution that enables Adobe Experience Manager (AEM) to have component content management solution (CCMS) capabilities for DITA-based content creation and delivery. Customers can access XML Documentation solution workflows programmatically using the XML Documentation solution APIs to integrate it with other enterprise applications. These APIs can also be used by Adobe partners to enhance the value proposition of XML Documentation solution by extending its functionality or by integrating it with other applications or services.

XML Documentation solution APIs

The XML Documentation solution APIs are available in two formats: HTTP and Java. These APIs expose key functions of XML Documentation solution to application developers. Using these functions, developers can create their own plug-ins to extend the out-of-the-box workflows. The APIs are available around managing outputs for DITA content, working with DITA maps, adding conditional attributes to folder-level profiles, and converting HTML and Words documents to DITA format.

Installing the JARs on your local Apache Maven repository

To be able to use the JAR files exposed by XML Documentation solution, you need to install them on your local Apache Maven repository. Perform the following steps to install the JARs on your location Maven repository:

1) Extract the contents of the XML Documentation solution package (`.zip`) file on your local system.
2) In the command prompt, navigate to the following folder in the extracted content path:
   `\jcr_root\libs\fmdita\osgi-bundles\install`
3) Run the following command to install the API bundle to your local Maven repository:
   ```
   mvn install:install-file -Dfile=api-3.2.jar -DgroupId=com.adobe.fmdita
   -DartifactId=api -Dversion=3.2 -Dpackaging=jar**
   ```

This process installs the API JARs in the local Maven repository.

REST APIs for Output management

The following REST APIs are available for managing output in XML Documentation solution.

Get all output presets for a DITA map

A POST method that retrieves all output presets configured for a DITA map.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/publishlistener

Parameters

Name	Type	Required	Description
`:operation`	String	Yes	Name of the operation being called. The value of this parameter is `getalloutputs`. **NOTE:** The value is case-insensitive.
`sourcePath`	String	Yes	Absolute path of the DITA map file.

Response values

Returns an array of JSON Output Preset objects, each object containing the following elements:

Element	Description
`outputName`	Name of the output preset. Output names are unique in the scope of the DITA map they are defined in.
`outputType`	Type of output generated using this preset, for example AEM Site, PDF, EPUB or other. The available options are: • AEMSITE • PDF • HTML5 • EPUB • CUSTOM
`outputTitle`	A descriptive name for the output preset settings. This is used to define the value for the Setting Name property for the output preset.
`ditaValPath`	Path of the DITAVAL file to be used to generate the desired output.
`targetPath`	Path where the output is published or stored.

Element	Description
`siteName`	*(For AEM Site output)* Name of the AEM Site.
`siteTitle`	*(For AEM Site output)* Title of the AEM Site.
`templatePath`	*(For AEM Site output)* Path of the template node to be used to generate the desired output.
`searchScope`	Specify the scope for the search operation. The value for this parameter must be set to `local`.
`generateTOC`	*(For AEM Site output)* Specify whether a TOC is generated (true) or not (false).
`generateBreadcrumbs`	*(For AEM Site output)* Specify whether the breadcrumbs are generated (true) or not (false).
`overwriteFiles`	*(For AEM Site output)* Specify whether files at the destination are overwritten (true) or not (false).
`pdfGenerator`	Specify the PDF generation engine to use. The possible values are: • DITAOT • FMPS

Create output preset

A POST method that creates a new output preset for a DITA map.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/publishlistener

Parameters

Name	Type	Required	Description
`:operation`	String	Yes	Name of the operation being called. The value of this parameter is `createoutput`. **NOTE:** The value is case-insensitive.
`sourcePath`	String	Yes	Absolute path of the DITA map file.

Name	Type	Required	Description
`outputTitle`	String	Yes	A descriptive name for the output preset settings. This is used to define the value for the Setting Name property for the output preset. **NOTE:** When a new output preset is created, the back-end system drives a unique name for the output preset from the given title.
`outputType`	String	Yes	Type of output generated using this preset, for example AEM Site, PDF, EPUB or other. The available options are: • AEMSITE • PDF • HTML5 • EPUB • CUSTOM

Response values

Element	Description
`outputName`	A unique name for the newly created output preset. This name is derived from the value of the `outputTitle` parameter.

Save output preset

A POST method that saves changes made in an output preset.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/publishlistener

Parameters

Name	Type	Required	Description
`:operation`	String	Yes	Name of the operation being called. The value of this parameter is `saveoutput`. **NOTE:** The value is case-insensitive.
`sourcePath`	String	Yes	Absolute path of the DITA map file.

Name	Type	Required	Description
`outputObj`	String	Yes	A JSON object that contains properties of the output preset that is being updated. The `outputObj.outputName` property contains the name of the output preset that is to be updated. For the format of the JSON object, see the *Response values* table in *Get all output presets for a DITA map*.

Response values

Returns a HTTP 200 (Successful) response.

Get a specific output preset

A POST method that retrieves an existing output preset.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/publishlistener

Parameters

Name	Type	Required	Description
`:operation`	String	Yes	Name of the operation being called. The value of this parameter is `getoutput`. **NOTE:** The value is case-insensitive.
`sourcePath`	String	Yes	Absolute path of the DITA map file.
`outputName`	String	Yes	Name of the output preset for which the details have to be retrieved.

Response values

Element	Description
`outputName`	Name of the output preset. Output names are unique in the scope of the DITA map they are defined in.

Element	Description
`outputType`	Type of output generated using this preset, for example AEM Site, PDF, EPUB or other. The available options are: • AEMSITE • PDF • HTML5 • EPUB • CUSTOM
`outputTitle`	A descriptive name for the output preset settings. This is used to define the value for the Setting Name property for the output preset.
`ditaValPath`	Path of the DITAVAL file to be used to generate the desired output.
`targetPath`	Path where the output is published or stored.
`siteName`	(For AEM Site output) Name of the AEM Site.
`siteTitle`	(For AEM Site output) Title of the AEM Site.
`templatePath`	(For AEM Site output) Path of the template node to be used to generate the desired output.
`searchScope`	Specify the scope for the search operation. The value for this parameter must be set to `local`.
`generateTOC`	(For AEM Site output) Specify whether a TOC is generated (true) or not (false).
`generateBreadcrumbs`	(For AEM Site output) Specify whether the breadcrumbs are generated (true) or not (false).
`overwriteFiles`	(For AEM Site output) Specify whether files at the destination are overwritten (true) or not (false).
`pdfGenerator`	Specify the PDF generation engine to use. The possible values are: • DITAOT • FMPS

Generate output

A GET method that generates output using one or more output presets.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/publishlistener

Parameters

Name	Type	Required	Description
`operation`	String	Yes	Name of the operation being called. The value of this parameter is `GENERATEOUTPUT`. **NOTE:** The value is case-sensitive.
`source`	String	Yes	Absolute path of the DITA map file.
`outputName`	String	Yes	Name of the output preset(s) to be used to generate output. Multiple output presets can be specified using a pipe (“|”) delimiter, for example `aemsite|pdfoutput`.

Response values

Returns a HTTP 200 (Successful) response.

Generate incremental output

A GET method that generates incremental output for an AEM Site using one or more output presets.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/publishlistener

Parameters

Name	Type	Required	Description
`operation`	String	Yes	Name of the operation being called. The value of this parameter is `INCREMENTALPUBLISH`. **NOTE:** The value is case-sensitive.

Name	Type	Required	Description

`contentPath`	JSON	Yes	Absolute path of the DITA map file and topic files along with the name of output presets. Use the following example as the building block:
			`{`
			`{`
			`"ditamap": "/content/dam/sample/sample.ditamap",`
			`"topics": [ "/content/dam/sample/topic1.xml", "/content/dam/sample/topic2.xml" ],`
			`"fullMaps": [ "/content/dam/sample/submap.ditamap" ],`
			`"maps": [ "/content/dam/sample/keyspace.ditamap" ],`
			`"outputs": [ "aemsite" ]`
			`}`
			`}`
			The `ditamap` attribute takes the absolute path of the DITA map that is used to generate the output.
			The `topics` attribute takes an array of topics that are updated and need to be republished.
			The `fullMaps` attribute contains path of the map files (like chunked submaps) that are needed along with their topics for incremental output generation.
			The `maps` attribute contains path of the map files (for resolving keyspace references) that are extracted on the disk without topics.
			The `outputs` attribute takes an array of output preset names that are used to generate the output.

Response values

Returns a HTTP 200 (Successful) response.

Delete output preset

A POST method that deletes an output preset.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/publishlistener

Parameters

Name	Type	Required	Description
`:operation`	String	Yes	Name of the operation being called. The value of this parameter is `deleteoutput`. **NOTE:** The value is case-insensitive.
`sourcePath`	String	Yes	Absolute path of the DITA map file.
`outputName`	String	Yes	Name of the output preset to delete.

Response values

Returns a HTTP 200 (Successful) response.

REST API to work with DITA maps

The following REST API allows you to work with DITA maps in XML Documentation solution.

Download DITA map with dependents

A GET method that downloads a DITA map with all its dependents such as referenced topics, sub-maps, images, and DTDs used in the map and topics.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/fmdita/exportditamap

Parameters

Name	Type	Required	Description
`ditamap`	String	Yes	Absolute path of the DITA map file in AEM repository.
`baseline`	String	Yes	The name of the baseline that is used to retrieve the versioned content. Every baseline in a DITA map has a unique internal name, which can be retrieved using the method. **NOTE:** The name displayed in the Baseline tab in the DITA map console is the baseline title and not baseline name.

Response values

A `.zip` file whose contents are written to the output stream of the response.

REST API to work with conditional attributes

The following REST API allows you to add conditional attributes in a folder profile.

Add conditional attribute in a folder-level profile

A POST method that adds conditional attributes to a given folder-level profile.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/fmdita/folderprofiles

Parameters

Name	Type	Required	Description
`:operation`	String	Yes	Name of the operation being called. The value of this parameter is `ADDATTRIBUTEPROFILES`. **NOTE:** The value is case-insensitive.
`profilename`	String	Yes	Display name of the folder-level profile in which the conditional attributes have to be added.
`conditional profiles`	JSON array	Yes	A JSON array consisting of the conditional attribute name and values. The following example code snippet shows the JSON array with two attributes - `platform` and `product` with multiple values assigned to them. `[ { name: "platform", values: [ {value: "win", label: "Windows"}, {value: "mac", label: "Mac OS"} ]},{ name: "product", values: [ {value: "aem_1", label: "AEM 6.1"}, {value: "aem_4, label: "AEM 6.4"} ] }]`

Response values

Returns a HTTP 200 (Successful) response.

REST APIs for conversion workflow

The following REST APIs allows you to convert Word, HTML, and InDesign documents into DITA format.

Convert Word documents

A GET method that converts Word documents into DITA format.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/fmdita/conversion

Parameters

Name	Type	Required	Description
`operation`	String	Yes	Name of the operation being called. The value of this parameter is `word2dita`. **NOTE:** The value is case-insensitive.
`inputFile`	String	Yes	Absolute path of the source Word files in AEM repository.
`destPath`	String	Yes	Absolute path of the destination location where the converted DITA files will be saved.
`createRev`	Boolean	Yes	Specify whether a revision of the files is created (`true`) at the specified destination or not (`false`). This is considered only when the destination location contains an existing version of the converted files.
`style2tagMap`	String	Yes	Absolute path of the style mapping file that will be used for conversion.

Response values

Returns a HTTP 200 (Successful) response.

Convert HTML documents

A GET method that converts HTML documents into DITA format.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/fmdita/conversion

Parameters

Name	Type	Required	Description
`operation`	String	Yes	Name of the operation being called. The value of this parameter is `html2dita`. **NOTE:** The value is case-insensitive.
`inputFile`	String	Yes	Absolute path of the source HTML files in AEM repository.
`destPath`	String	Yes	Absolute path of the destination location where the converted DITA files will be saved.
`createRev`	Boolean	Yes	Specify whether a revision of the files is created (`true`) at the specified destination or not (`false`). This is considered only when the destination location contains an existing version of the converted files.

Response values

Returns a HTTP 200 (Successful) response.

Convert InDesign documents

A GET method that converts InDesign documents into DITA format.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/fmdita/conversion

Parameters

Name	Type	Required	Description
`operation`	String	Yes	Name of the operation being called. The value of this parameter is `idml2dita`. **NOTE:** The value is case-insensitive.
`inputFile`	String	Yes	Absolute path of the source InDesign files in AEM repository.
`destPath`	String	Yes	Absolute path of the destination location where the converted DITA files will be saved.
`createRev`	Boolean	Yes	Specify whether a revision of the files is created (`true`) at the specified destination or not (`false`). This is considered only when the destination location contains an existing version of the converted files.

Response values

Returns a HTTP 200 (Successful) response.

REST API for creating and activating packages

The following REST API allows you to create and activate CRX packages.

Create and activate package

A POST method that creates and activates CRX package.

Request URL

http:// *<xml-documentation-server>*: *<port-number>*/bin/fmdita/activate

Parameters

The request query consists of the JSON rules string. The content type of the POST request must be set to `application/json; charset=UTF-8`.

Example

The following examples shows the API call using the curl command:

```
curl -u <username>:<password> -H "Content-Type: application/json;
charset=UTF-8"  -k -X POST -d "{JSON rules string}"
http://<xml-documentation-server>:<port-number>/bin/fmdita/activate
```

Java-based API for working with output generation

The following Java-based API allows you to generate output for a DITA map. This API is available in the form of a bundle. You must include this bundle in your code to use this APIs.

Bundle details:

- Group ID: **com.adobe.fmdita**
- Artifact ID: **api**
- Version: **3.4**
- Package: **com.adobe.fmdita.api.maps**
- Class details:

  `public class` **`PublishUtils`** `extends Object`

  The ***`PublishUtils`*** *class contains a method for generating output for one or more output presets.*

Generate output

The `generateOutput` method generates output for a DITA map file using the specified output presets.

Syntax

```
public static void generateOutput(Session session,
                  String sourcePath,
                  String outputName)
                  throws RepositoryException, IOException, Exception
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.
`sourcePath`	String	Path (in the AEM repository) of the DITA map file for which the output needs to be generated.
`outputName`	String	Name of the output preset(s) to be used to generate output. Multiple output presets can be specified using a pipe (“|”) delimiter, for example `aemsite|pdfoutput`.

Exception

Throws `javax.jcr.RepositoryException`, `java.io.IOException`, and `java.lang.Exception`.

Java-based APIs to work with DITA maps

The following Java-based APIs allow you to work with DITA maps in XML Documentation solution. These APIs are available in the form of a bundle. You must include this bundle in your code to use these APIs.

Bundle details:

- Group ID: **com.adobe.fmdita**
- Artifact ID: **api**
- Version: **3.2**
- Package: **com.adobe.fmdita.api.maps**
- Class details:

  `public class` **`MapUtilities`** `extends Object`

  The MapUtilities class contain methods for retrieving metadata information from a DITA map file.

Download DITA map with dependents

The `zipMapWithDependents` method creates a `.zip` file containing a DITA map along with all its dependents such as referenced topics, sub-maps, images, and DTDs. The `.zip` file for the DITA map is created based on a given baseline.

It also allows you to either maintain the same structure (parent and child folders) or create a flat file structure within a single folder for all dependent files.

IMPORTANT: *The API will throw an exception and fail to create a* `.zip` *file if any of the dependent files are missing.*

Syntax

```
public static void zipMapWithDependents(Session session,
                      String sourcePath,
                      String baseline,
                      OutputStream outputStream,
                      boolean flatFS)
                      throws RepositoryException, IOException
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.
`sourcePath`	String	Path (in the AEM repository) of the DITA map file that needs to be downloaded.
`outputStream`	java.io.OutputStream	The stream to write the ZIP to.

Name	Type	Description
`baseline`	String	The name of the baseline that is used to retrieve the versioned content. Every baseline in a DITA map has a unique internal name, which can be retrieved using the `getBaselineList` method. **NOTE:** The name displayed in the Baseline tab in the DITA map console is the baseline title and not baseline name.
`flatFS`	Boolean	(Optional) If set to `true`, then a flat structure of files is returned in the ZIP file. For example, if your DITA map refers to content in multiple folders, then all referenced files are pulled into a single folder. In case there are files with same name, then those files are renamed by adding a numeric suffix. All references (in DITA map and topics) are handled automatically, as they are updated based on the new location of files in the flat folder structure. If set to `false`, then the folder structure is maintained as is in the ZIP file. If the DITA map refers to files from multiple locations, then all those locations are also created in the ZIP file. When you restore the ZIP file, the exact folder structure is created at the destination location. The default value for this parameter is `false`.

Returns

Contents of the ZIP are written to the `outputStream`.

Exception

Throws `javax.jcr.RepositoryException`, `java.io.IOException`.

Get a list of baselines

The `getBaselineList` method retrieves a list of all baselines that exist for a given DITA map.

Syntax

```
public static List<HashMap<String,String>> getBaselineList(
                  javax.jcr.Session session,
                  String sourcePath)
                  throws javax.jcr.RepositoryException
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.
`sourcePath`	String	Path (in the AEM repository) of the DITA map file for which the baseline information is to be retrieved.

Returns

A list of `HashMap` objects. Each `HashMap` object represents a baseline and contains the name and title of the baseline.

Exception

Throws `javax.jcr.RepositoryException`.

Get a list of conditional presets

The `getConditionalPresetList` method retrieves a list of all conditional presets that exist for a given DITA map.

Syntax

```
public static List<HashMap<String,String>> getConditionalPresetList (
                  javax.jcr.Session session,
                  String sourcePath)
                  throws javax.jcr.RepositoryException
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.
`sourcePath`	String	Path (in the AEM repository) of the DITA map file for which the conditional preset information is to be retrieved.

Returns

A list of `HashMap` objects. Each `HashMap` object represents a conditional preset and contains the name and title of the conditional preset.

Exception

Throws `javax.jcr.RepositoryException`.

Get the DITAVAL file information for a conditional preset

The `getDitavalFromConditionalPreset` method retrieves the path of the DITAVAL file corresponding to a conditional preset for a given DITA map.

Syntax

```
public static String getDitavalFromConditionalPreset(
                  javax.jcr.Session session,
                  String sourcePath,
                  String cpName)
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.
`sourcePath`	String	Path (in the AEM repository) of the DITA map file for which the DITAVAL file is to be retrieved.
`cpName`	String	Name of the conditional preset in the DITA map for which the DITAVAL file is to be retrieved.

Returns

The path of the DITAVAL file corresponding to the conditional preset defined in the DITA map file.

Get all dependencies for a node

The `getAllDependencies` method returns all dependencies of a given node.

Syntax

```
public static List<Node> getAllDependencies(javax.jcr.Node rootNode)
```

Parameters

Name	Type	Description
`rootNode`	javax.jcr.Node	The root node for which all dependencies are to be retrieved.

Returns

A node list containing all dependencies of the root node.

Java-based APIs for conversion workflow

The following Java-based APIs allow you to convert HTML and Word documents into DITA format. These APIs are available in the form of a bundle. You must include this bundle in your code to use these APIs.

Bundle details:

- Group ID: **com.adobe.fmdita**
- Artifact ID: **api**
- Version: **3.2**
- Package: **com.adobe.fmdita.api.conversion**
- Class details:

  `public class` **`ConversionUtils`** `extends Object`

  *The **ConversionUtils** class contain methods for converting HTML and Word documents into DITA format.*

Convert HTML documents

The `convertHtmlToDita` method converts HTML documents into DITA format.

Syntax

```
public static void convertHtmlToDita(Session session,
                  String inputFile,
                  String destPath,
                  boolean createRev)
                  throws RepositoryException, WorkflowException
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.
`inputFile`	String	Absolute path of the source HTML files in AEM repository.
`destPath`	String	Absolute path of the destination location where the converted DITA files will be saved.
`createRev`	Boolean	Specify whether a revision of the files is created (`true`) at the specified destination or not (`false`). This is considered only when the destination location contains an existing version of the converted files.

Exception

Throws `RepositoryException`.

Convert Word documents

The `convertWordToDita` method converts Word documents into DITA format.

Syntax

```
public static void convertWordToDita(Session session,
                  String inputFile,
                  String destPath,
                  String style2tagMap,
                  boolean createRev)
                  throws RepositoryException, WorkflowException
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.
`inputFile`	String	Absolute path of the source Word files in AEM repository.
`destPath`	String	Absolute path of the destination location where the converted DITA files will be saved.
`style2tagMap`	String	Absolute path of the style mapping file that will be used for conversion.
`createRev`	Boolean	Specify whether a revision of the files is created (`true`) at the specified destination or not (`false`). This is considered only when the destination location contains an existing version of the converted files.

Exception

Throws `RepositoryException`.

Java-based APIs for creating baseline and working with labels

The following Java-based APIs allow you to create baseline and add labels to files in a baseline. These APIs are available in the form of a bundle. You must include this bundle in your code to use these APIs.

Bundle details:

- Group ID: **com.adobe.fmdita**
- Artifact ID: **api**
- Version: **3.2**
- Package: **com.adobe.fmdita.api.maps**
- Class details:

  `public class BaselineUtils extends Object`

  *The **BaselineUtils** class contain methods for creating baselines and applying labels to files in a baseline.*

Create a baseline

The `createBaseline` method creates a baseline as on a given date and time. On successful creation of the baseline, it returns the name of the baseline.

Syntax

```
public static String createBaseline(Session session,
                  String sourcePath,
                  String baselineTitle,
                  Date versionDate)
                throws RepositoryException, WorkflowException, Exception
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session. The user session needs to have both read and write permissions for the DITA map and read permission for all the reference files included in the baseline.
`sourcePath`	String	Absolute path of the DITA map file in AEM repository.
`baselineTitle`	String	A unique title for the baseline.

Name	Type	Description
`versionDate`	Date	The baseline is created using the versions of topics (directly referenced from the DITA map) as on this date. Specify the date in `d-MM-yyyy H:mm` format.

Returns

The name of the baseline, which is the node name of the baseline in the JCR repository. The title of the newly created baseline will be shown to the user on the Baseline page for the DITA map.

Exception

Throws `RepositoryException`.

Apply labels

The `applyLabel` method applies one or multiple labels to the files in a baseline.

Syntax

```
public static void applyLabel(Session session,
                  String sourcePath,
                  String baselineName,
                  String label)
                throws RepositoryException, WorkflowException, Exception
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.
`sourcePath`	String	Absolute path of the DITA map file in AEM repository.
`baselineName`	String	Name of the baseline node on which the label has to be applied. To get the name of the baseline node, you can use the method or check the `baselines` node of the DITA map in CRXDE. **NOTE:** Label is applied to version of files that are directly referenced from the map file in the baseline.
`label`	String	A label that is applied on files in the baseline. Ensure that the label does not contain the following characters: `/,:,[,],|,*` In case you want to set multiple labels, then separate labels with a comma; for example `Label1, Label2`.

Exception

Throws `RepositoryException`.

Delete labels

The `deleteLabel` method deletes one or multiple labels from the files in a baseline.

Syntax

```
public static Map<String, String> deleteLabel(
                      Session session,
                      String sourcePath,
                      String baselineName,
                      String label)
                    throws RepositoryException, VersionException, Exception
```

Parameters

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.
`sourcePath`	String	Absolute path of the DITA map file in AEM repository.
`baselineName`	String	Name of the baseline from which the label has to be deleted. **NOTE:** Label is deleted from the version of files that are directly referenced from the map file in the baseline.
`label`	String	A label that is to be deleted from files in the baseline. In case you want to delete multiple labels, then separate labels with a comma; for example `Label1, Label2`.

Returns

The map with *key:value* pair of `path:deletedlabels` for all files in the baseline.

Exception

Throws `RepositoryException, VersionException, Exception`.

Java-based API for working with folder profiles

The following Java-based API allows you to add conditional attributes to a folder-level profile. This API is available in the form of a bundle. You must include this bundle in your code to use this APIs.

Bundle details:

- Group ID: **com.adobe.fmdita**
- Artifact ID: **api**
- Version: **3.2**
- Package: **com.adobe.fmdita.api.profiles**
- Class details:

  `public class` **`FolderProfileUtils`** `extends Object`

  The ***`FolderProfileUtils`*** *class contains a method for adding conditional attributes in a folder profile.*

Add conditional attributes to a folder profile

The `addAttributeProfiles` method adds conditional attributes to a folder-level profile.

Syntax

```
public static boolean addAttributeProfiles(
                  List<String> attributeNames,
                  List<String> values,
                  List<String> labels,
                  String profileName,
                  Session session)
                  throws Exception
```

Parameters

Name	Type	Description
`attributeNames`	String	A list of attribute names.
`values`	String	A list of values for the given attributes.
`labels`	String	A list of labels for the **`<attribute>`**- **`<value>`** pairs. [1]
`profileName`	String	The name of the folder-level profile to which these attributes, values, and labels have to be applied. **IMPORTANT:** All existing attributes-values-labels defined in the profile are overwritten.

Name	Type	Description
`session`	javax.jcr.Session	A valid JCR session.

1. The `attributeNames`, `values`, and `labels` at the same index in an array list must correspond to the same entry.

Returns

`true` for success. In case of a failure, it throws an exception.

Exception

Throws `java.lang.Exception` in the following scenarios:

- If the API couldn't get `resourceResolverFactory` object. In that case, you should restart the bundle.
- If parameters passed to the API are invalid.
- If the API is called through unauthorized user session, such as the user who is not an administrator for the given folder profile.

Java-based API for creating and activating packages

The following Java-based API allows you to create and activate CRX packages. This API is available in the form of a bundle. You must include this bundle in your code to use this APIs.

Bundle details:

- Group ID: **com.adobe.fmdita**
- Artifact ID: **api**
- Version: **3.3**
- Package: **com.adobe.fmdita.api.crxactivate**
- Class details:

  `public class` **`CRXActivator`**

  The ***`CRXActivator`*** *class contains a method for creating CRX packages and replicating it on the publish instance.*

Create and activate packages

The `activate` method creates a CRX package on the author instance and replicates it on the publish instance, if required. It is assumed that the AEM replication parameters have already been setup on the author instance. This method creates the CRX package based on a list of rules provided as input parameters in a JSON string.

NOTE: Errors encountered during the creation or activation process are written to the `outputstream`.

Syntax

```
public static void activate(String json,
                     OutputStream outputstream,
                     Session session)
                     throws Exception
```

Parameters

Name	Type	Description
`json`	String	JSON string that determines the CRX package to be built. Use the following format to create the JSON string: • `activate`: Is of type Boolean (`true`/`false`). Determines whether the CRX package created in the author instance is replicated to the publish instance. • `rules`: Is of type JSON Array. An array of JSON rules which are processed sequentially to build the CRX package. • `rootPath`: Is of type String. The base path upon which the node/property queries are executed. If no node/property queries are present, the root path and all nodes present under the root path are included in the CRX package. • `nodeQueries`: Is of type Regex Array. An array of regular expressions used to include specific files under the root path. • `propertyQueries`: Is of type JSON Array. An array of JSON Objects with each JSON Object consisting of an XPath query to be executed on the root path and the name of a property present in each JCR node after the query is executed. The value of the property in each JCR node should be a path or an array of paths. The paths present in this property are added to the CRX package.
`outputstream`	java.io.OutputStream	This is used to write the result of various stages, such as query execution, file inclusion, CRX package creation, or activation. Any error encountered during creation or activation process are written to the `outputstream`. This is useful for debugging.
`session`	String	A valid JCR session with activation permission.

Exception

Throws `java.io.IOException`.

Example

The following example shows how to build a JSON query:

```
{
  "activate": true,
  "rules": [
```

```
    {
      "rootPath": "/content/dam/nested",
      "nodeQueries": [
        ".*\\.jpg",
        ".*\\.png",
        ".*\\.gif"
      ]
    },
    {
      "rootPath": "/content/output/sites/hierarchy_ditamap"
    },
    {
      "rootPath": "/content/output/sites/hierarchy_ditamap",
      "propertyQueries": [
        {
          "query": "//*[@fileReference]",
          "property": "fileReference"
        }
      ]
    }
  ]
}
```

The example JSON query consists of the following rules:

- Only the `.png`, `.jpg`, and `.gif` images under `/content/dam/nested` path are included in the package.
- All node under `/content/output/sites/hierarchy_ditamap` are included in the package.
- The paths present in the `fileReference` property of nodes under `/content/output/sites/hierarchy_ditamap` are included in the package.

Post-processing event handler

XML Documentation solution exposes `com/adobe/fmdita/postprocess/complete` event that is used to perform any post-processing operations. This event is triggered whenever an operation is performed on a DITA file. The following operations on a DITA file trigger this event:

- Upload
- Creation
- Modification
- Deletion

You need to create an AEM event handler to read the properties available in this event and do further processing.

Event details are explained below:

Event name:

`com/adobe/fmdita/postprocess/complete`

Parameters

Name	Type	Description
`path`	String	The path of the file that triggered this event. Typically, this is the file on which an operation has been performed.
`status`	String	The return status for the operation performed. The possible options are: • `SUCCESS`: The post-processing operation completed successfully. • `COMPLETED WITH ERRORS`: The post-processing operation completed, but with some errors. • `FAILED`: The post-processing operation failed due to some fatal error.
`message`	String	In case the `status` is `COMPLETED WITH ERRORS` or `FAILED`, this parameter contains the details about the error or the reason of failure.
`operation`	String	The post-processing operation performed on the file. The possible options are: • `Addition` • `Updation` • `Deletion`

NOTE: This event is not triggered for the Deletion operation in AEM 6.1.

Conversion process event handler

XML Documentation solution exposes `com/adobe/fmdita/conversion/complete` event that is used to perform any post-processing operations after completion of a document conversion process. This event is triggered whenever an non-DITA document is migrated into DITA file format. For example, if you run a Word to DITA conversion or InDesign to DITA conversion process, this event is called after the conversion process ends.

You need to create an AEM event handler to read the properties available in this event and do further processing.

Event details are explained below:

Event name:

`com/adobe/fmdita/conversion/complete`

Parameters

Name	Type	Description
`status`	String	The return status for the operation performed. The possible options are: • `SUCCESS`: The conversion process completed successfully. • `COMPLETED WITH ERRORS`: The conversion process completed, but with some errors. • `FAILED`: The conversion process failed due to some fatal error.
`filePath`	String	Absolute path of the source file (to be converted) in AEM repository.
`outputPath`	String	Absolute path of the destination location where the converted DITA files will be saved.
`logPath`	String	Absolute path of the node where the conversion log will be saved.