Realistic Example and Hands-on Step-by-Step Guide for WCM Multi-locale Delivery

Introduction

Objective

IBM WCM Multilingual framework is already available, and no better WCM Multi-locale delivery approach can raise the bar for its consistency. However, it can be challenging for WCM developers to apply the framework to real-life use cases or understand what framework components they should use.

The attached document is a concise piece that focuses on a realistic example offering step-by-step hands-on experience for designers and developers, to show how they can apply MLS to a realistic business use case from the ground up. The document will also cover additional localization add-ons that can be used for the overall application release within Portal i.e. Text Providers, Locale Plugin, Dot Notation, 'Current Content' / 'Current SiteArea' options in Menus and Navigators, and Creating Localized URLs for Portal Pages.

Document Scope

This document provides a detailed understanding for a Macro Design of WCM technical artifacts and components, related to WCM Multi-locale delivery. The following sections will identify the fields and details needed for libraries, workflows, authoring templates, rendering components, syndication, and administration.

History

When our clients refer to localization, it does not necessary mean the multi-locale support only, because WebSphere Portal, for example, comes with 31 languages out-of-the box and there are additional language packs available from local partners. Integrated components (for example Connections portlets...etc) are also available in the commonly required languages. WebSphere Portal framework offers the capability to make every added component available in multiple languages.

However, what it is equally important for our clients is the multilingual support for clients' "contents", and how smooth for data entry personnel to use the product to issue moderated multilingual contents in no time.

Implementing multilingual applications/modules hosted in IBM WebSphere Portal via Web Content Manager has been a mandatory requirement for almost every client I have worked with. Starting with IBM Workplace Server 2.5, WebSphere Portal 5.0 and up to the recent version 8.0.0.1, there have been many bespoke approaches and techniques to accommodate such crucial request for most of our clients.

Localization requirements and how smooth our ICS portfolio delivers them, versus, other competitive product's approaches, is surely one of the vital spots that would make a stratified client out of our work.

Having said that, it is not all about if your product eventually "can" provide a "framework" for clients' developers, web managers, data entry personnel, and administrators; but rather, how complex your framework is and what kind of skills the client must secure for such roles. The more complex your framework/approach is, the more the client gets skeptical about why they should go with your product.

Throughout the years, there has been many approaches to provide multilingual support for contents via IBM Web Content Management, for example:

- In 2008, a developerWorks article (Implementing multilingual sites using IBM Workplace Web Content Management in IBM WebSphere Portal) was published. It was great, considering the limitations of IBM Workplace Web Content Management back then. It uses bespoke development in WCM using JSP components together with Portal's Personalization Engines to develop/deliver personalized English/Arabic contents. It also uses an Authoring Template for each locale delivered and a replica of the Site Structure of each locale. Again it was a great approach, and basically the only one available, but it requires a great deal of development skills for developers, mandate the need for IT people for maintenance, and obviously error-prone for client non-IT personnel.

- The previous approach has been (and still) used, some minor changes may apply though. Instead of using different Authoring templates for each locale, use a single template for both locales that has its elements multiplied, one for each locale.

- In 2010, a developerWorks article (Multi-locale site management with IBM Workplace Web Content Management) was published by a WCM Development Lab in Sydney, which was a greater step ahead for what could be a solid framework for Multilingual content support in WCM. The approach has been a core for any upcoming versions of MLS until it is out-of-the-box plugin in Portal. In 2010 though, all the custom developed plugins for this framework has been shared in various sample JSP codes that the developer must inject, and the approach was not formally supported by IBM just yet.

- Later IBM offered a supported version of such framework in a free packaged extensions that can be downloaded from IBM WebSphere Portal Business Solutions Catalogue (Greenhouse). It's called (IBM Lotus Web Content Management Multilingual Solution). The extension has scripts to install/uninstall the framework components automatically with minimal configuration steps by the developers.

- Lastly a refined version of the framework has been integrated out-of-the-box in WebSphere Portal V8.0.0.1.

Client Requirements

Our imaginary client might typically has the following requirements:

- Website must support, for example, Arabic and English delivery for each content, with English as the base language.

- Website renders the appropriate content in a language that corresponds to the selected locale.
- Authoring forms must also support Arabic and English labels that match author's selected locale.
- Translators should be notified when a content with a base language is ready for translation.
- Contents do not get published until the defined publish date is due.
- Contents All contents versions (with different languages) get published at the same time.
- All contents versions (with different languages) get expired at the same time.
- Once expired, contents are scheduled to be permanently deleted from the system. No soft deletion. All versions are deleted accordingly.

Library Strategy

WCM content and artifacts are according to best practice propagated to different environments using a syndication process. XMLAccess based deployments of WCM artifacts are not possible.

Transferring the web content library to another server will result in transferring test content items together with design artifacts. In order to overcome this issue web content data will divided into 2 libraries.

One library which is the “Design” Library will hold all WCM artifacts representing the design such as menus, presentation templates, authoring templates and categories. This is called "*DesignLibrary*"

The other library which is the “Content” Library will hold the site structure and the contents. The site structure is basically Site areas. The separation of artifacts and content into 2 separate libraries will allow deployment of only the design items by only transferring the “*DesignLibrary*" Library.

Two content libraries are presented for each supported locales (Arabic and English) i.e. "*ContenLibraryAR*" and "*ContenLibraryEN*". Note that you can export any library using the out of the box WCM library export functionality to produce backups that can be version controlled.

Design Library

Transferring of the “*DesignLibrary*" Library can be done using WCM out of the box syndication functionality as this will work even when transferring the library to an environment which already had the library transferred to before, what will actually happen is that only updates will be transferred (syndicated).

Content Libraries

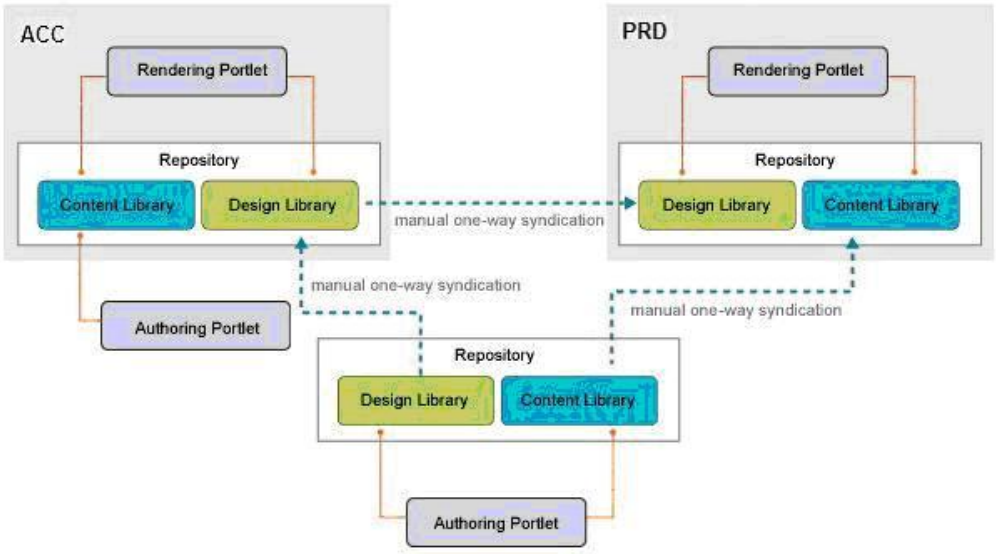
Both "*ContentLibraryAR* " and "*ContentLibraryEN*" content Libraries will include the site area structure, default content item(s) and user data contents...etc . You will only create the base library *ContentLibraryEN* with all the site structure required, then you will use the built-in Library Copy Portlet that is shipped with IBM MLS Solution to create the other localized Arabic library *ContentLibraryAR*.

The transfer of the content Libraries will be done using WCM out of the box library syndication functionality.

WCM Environments

Typically, there are three basic environment scenarios for IBM Web Content Manager; authoring, staging and delivery.

1. The Authoring Server: An authoring server is used to create and manage Web Content. Approved contents will be directly syndicated to the Production (Delivery) Environment.
2. The Staging Server (Acceptance): Although you can syndicate data directly from an authoring server to a delivery server, like the case of the approved contents, there are occasions where it is beneficial to include a staging (Acceptance) server in your system. This server where the Design items will be syndicated to and with another authoring portlet, users will be able to validate the new design for the published contents. After approval, design items will be syndicated to the delivery server
3. The Delivery Server (Production): A delivery server is used to display Web content to end-users using the local rendering portlets. The delivery file will receive syndicated contents from the authoring server and syndicated design items from the staging server.



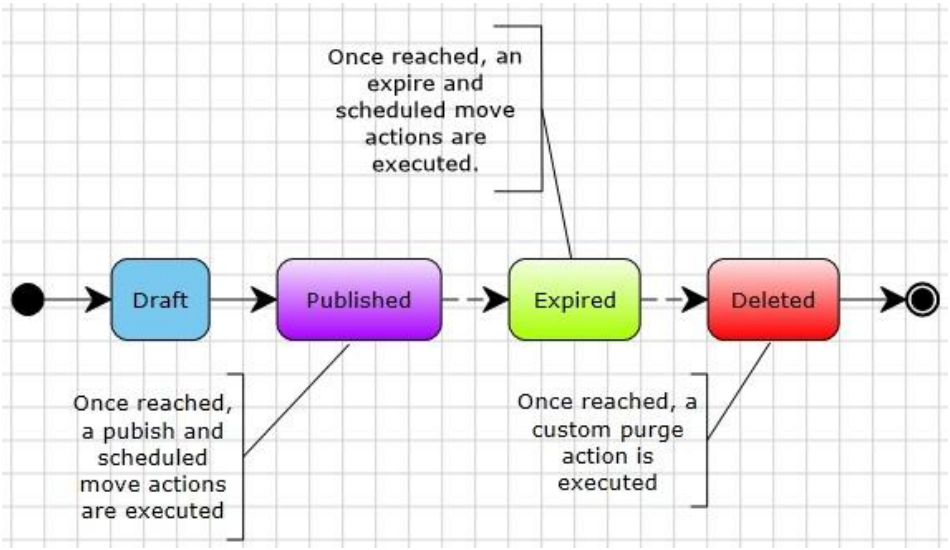
Refer for the "Best Practices for using IBM Workplace Web Content Management V6.0" document for the Do's and Don'ts regarding the deployment and syndication.

Workflow Strategy

There are three status levels during a workflow; Draft, Published, or Expired. Only published contents are available on the rendered site. No matter how many workflow stages are created, the content yet to be marked as one of these status levels.

This status is not linked to any stage, but is determined by actions. For example, a Publish action will change the status of a document from Draft to Published, if and only if the optional publish date is due.

The workflow for contents of our requirements is presented in the below state machine diagram.



Workflow Description

The following describes the different stages subjected to all contents along their life cycle through the above workflow.

- An author creates new content and saves it in his workspace. The content will be in the Draft stage (Draft status), and available for further editing by the relevant Content Authors group.
- Once the author is satisfied with the content, he approves it to move it to the Published stage.
- A Publish action is triggered, however the content will be in "Pending Published" state until the publish date is due; this is if a published date optionally specified while authoring the content, otherwise it goes Published and rendered in the website immediately.
- At the same time, a Scheduled Move action will be triggered, at the "Published" stage entrance. The action trigger is the Expiry date specified by the Author. If no expiry date specified, the item moves immediately to expiry, so check "Do not run the action if date has already been reached"

to give authors the flexibility to not providing any expiration for contents to stay live forever, until manually deleted.

- If an expiry date specified, and it is due, the content automatically moves to the Expiry stage, where an expire action is triggered to change the status to Expired, and the content will be removed from the rendering site.
- At the same time, another Scheduled Move will be triggered at the "Expired" stage entrance. The action trigger is the (General Date One) date specified by the Author. If no (General Date One) specified, the item moves immediately for deletion, so check "Do not run the action if date has already been reached" to give authors the flexibility to not providing any deletion due for contents to only remain expired, until manually deleted or republished.
- As per our requirements, it is required to totally "purge" contents once they get reach the "Deleted" stage. A custom action will be assigned at the entrance of the "Deleted" workflow stage. This custom action, called "Purge", will completely execute unrecoverable content deletion.
- Users assigned an "Approve" clearance of the Publish stage can manually moves the content to the Expiry Stage, but it won't be expired until the expiry date is due (if optionally specified while creating contents); same applies for a forced move to the "Deleted" stage.

The following table illustrates the processes, stages, actions, and status that describe the applied workflow. (Note that all action are to be configured as they enter the stage)

Process	Stage	Status Level
A content is first created and saved	Stage 1, Draft	Draft
If approved by the author, the content would be moved to the next stage. Two actions are triggered but pending at this content, the Publish and Scheduled Move actions The Status is Draft Pending Published. This is because the time specified in the Publish Date field of the content has not yet been reached.	Stage 2, Published	Draft Pending Published
Once the Publish Date is reached (or it was left empty), the content's Status changes to Published The item is still in (Stage 2, Published), only its	Stage 2, Published	Published

Status has changed.		
When the Expire Date is reached, the content is automatically moved to the next stage. As both the Scheduled Move and Expire Actions are using the Expired date, the content's status immediately changes to Expired.	Stage 3, Expired	Expired
Another Scheduled Move action is triggered herein, against a defined "General Date One" which moves the content to the next stage	Stage 3, Expired	Expired
Once the content reaches the Deleted Stage, a custom "Purge" action is trigged and completely removes the content from WCM repository	Stage 4, Deleted	Purged

There will be additional workflow stage called "Localize", added right after the "Draft" stage to automatically copy the content to the Arabic Library to get translated. Also, a synchronized publishing mechanism will be enabled to make sure both copies get published at the same time. Both features are discussed in later sections.

Workflow Security Settings

As the contents are participating in workflow, the content author is given delete access to the item only in the Draft stage. As the content progresses through the workflow, the content security is determined by the combined workflow and system defined security.

Users and groups can also be given different access levels to an item in a workflow stage. (These are specified in the properties section of the workflow stage, not the security section.) Workflow security uses the same levels as item security (Read, Edit, and Delete) with the addition of "Approve".

"Approve" authorizes the user/group to:

- Approve an item within a workflow.
- Create drafts of published items.

- View an item in the Authoring Portlet.
- View an item in the rendered Web site.

The following table shows the security levels granted for some assumed groups as the content propagates in its workflow defined above.

Note:

- If SiteAmins decide to delete non-draft content manually, it will be s soft deletion. The content must reach to the "Deleted" stage for the new custom action (Purge) to execute.
- Content Authors are granted Edit access while contents are published, to leverage the use of In-line Editing for ease of use.
- It does not matter what permissions to be granted in the "Deleted" stage, as contents are purged permanently upon arrival.

	Draft	Published	Expired
Read	[all users]		
Edit	ContentAuthors		
Delete	ContentAuthors SiteAmins	SiteAmins	SiteAmins
Approve	ContentAuthors SiteAmins	ContentAuthors SiteAmins	SiteAmins

Purge Action

WCM indeed provides some useful built in actions, which we have used most of them in the workflow strategy previously explained. A custom action can be used to execute a custom workflow action based on a Java class you have previously created and added to your system.

This gives our client the flexibility to adapt workflow behavior according to organizational changes with new policies which would eventually affect the deployed workflow. A new custom action can be triggered at the beginning or the end of the workflow stage of concern. For example, you can create new custom action that notifies the program managers for the rejection of content through SMS instead of an email!

By default, WCM performs soft-delete operation for non-draft documents (i.e. contents). Draft contents are always permanently deleted. There is only one custom action, that "purges" or " permanently deletes" expired contents for good. The following steps show how you create the "Purge" custom action our client uses in the Deleted stage:

1. In RSA, create New Dynamic Web Project.

2. Create a new Java Class which implements *com.ibm.workplace.wcm.api.custom.CustomWorkflowAction*
3. Implement execute method of *CustomWorkflowAction* . and get *WebContentCustomWorkflowService* object by using JNDI Lookup
4. Create new java factory class which implements *com.ibm.workplace.wcm.api.custom.CustomWorkflowActionFactory*
5. Create new *plugin.xml* under *WEB-INF*
6. Make starting weight of the application as same as starting weight of WCM application; by default it should be 20.
7. Deploy application; a New custom workflow action will be available in WCM

A sample code of the "Purge" custom action would be

// OurClientPurgeAction.java

```
public class OurClientPurgeAction implements
CustomWorkflowAction {

public CustomWorkflowActionResult execute(Document
document) {

WebContentCustomWorkflowService  wfService = null;

CustomWorkflowActionResult _results = null;

Directive directive = Directives.DELETE;

//Without the following line, it would softly delete the
document

((DeleteDirectiveParams)directive.createDirectiveParams()).
setPermanentDelete(true);

try {

   InitialContext ctx = new InitialContext();

   // Retrieve WebContentCustomWorkflowService using
JNDI name

   wfService = (WebContentCustomWorkflowService)
ctx.lookup("portal:service/wcm/WebContentCustomWorkflow
Service");

   _results = wfService.createResult(directive, "Our client
document is permanently deleted ");

 } catch(Exception e) {

e.printStackTrace();

}
```

// OurClientPurgeFactory.java

```
public class OurClientPurgeFactory implements
CustomWorkflowActionFactory {

public CustomWorkflowAction getAction(String arg0,
Document arg1) {

return new OurClientPurgeAction();

}

public String getActionDescription(Locale arg0, String
arg1) {

return " Permanently deletes our client documents " ;

}

public String[] getActionNames() {

return { "Delete document"};

}

public String getActionTitle(Locale arg0, String arg1) {

return "OurClientPurgeActionID";

}

public String getName() {

return " OurClientPurgeActionID " ;

}

public String getTitle(Locale arg0) {

return "Purge ;
```

```
return _results;

}

public Date getExecuteDate(Document document) {

// Execute immediately

return new Date();

}

}
```

```
}

}
```

// plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<plugin id=" com.ibm.ourclient.workplace.wcm.api.custom"

name="Purge Custom Workflow Action Factory"

version="1.0.0"

provider-name="IBM">

<extension

point="com.ibm.workplace.wcm.api.CustomWorkflowActionFactory"

id="OurClientPurgeActionID">

<provider class="your.package.name.wcm.OurClientPurgeFactory"/>

</extension>

</plugin>
```

More about creating custom actions; please check at: Creating a custom workflow action class

Localization Strategy

IBM Web Content Multilingual Solution (MLS)

The IBM Web Content Management Multilingual Solution (MLS) offering comprises a set of supported extensions to IBM Web Content Manager (WCM) that make it easier to build, maintain and deliver WCM sites in multiple languages. The MLS offering also provides documentation that describes the framework

for managing a multilingual site, the deployment and configuration of the extensions, and examples of their use.

You get find the MLS from Greehouse. Starting WCM 8.0.0.1 and onwards, the Multilingual Solution is available out of the box. Please check the official documentation on how to enable it: WCMv8 Documentation - MLS

The MLS package consists of the following extensions:

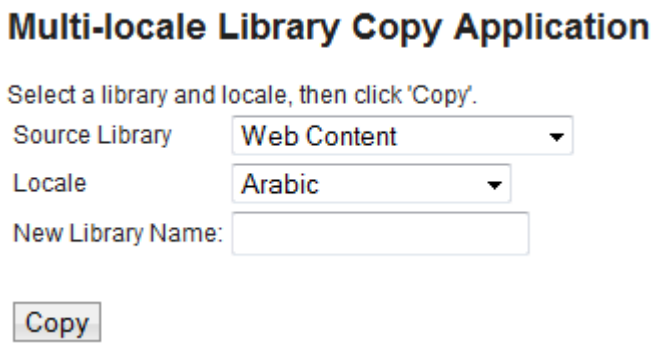
- Authoring Plugin (aka Custom Field) to display information about associated translations
- Workflow Synchronization Plugins (aka Custom Workflow Actions) to automate creation of translated documents and handle publishing, expiry and delete synchronization
- Rendering Plugins (aka LRP Context Processor and Servlet JSPs) to able end-users to see content in the right language and allow switching between languages
- Library Copy Portlet to make it easy to roll out new translated versions of existing sites.

Prerequisites to use MLS

- One library should be created for the base site (*ContentLibraryEN*), another for shared templates and components (*DesignLibrary)*, and then one library for each localized site (*ContentLibraryAR*).
- For content to be recognized as being equivalent, it must have the same path in each localized library (display names can be different of course). For example, only the library name will be different for content displayed in English and Arabic localities:
    - http://host/wps/wcm/connect/contentlibraryen/sitearea/content
    - http://host/wps/wcm/connect/contentlibraryar/sitearea/content
    - Normally you would have created the *ContentLibraryEN* library with its site structure, and use the "Library Copy Portlet" to copy it renamed by *ContentLibraryAR*, along with setting its locale to Arabic
- Creating, updating, publishing, expiring, deleting, and even moving content should be performed using workflow.
- Presentation templates and components, should all be shared across locales as much as possible, however they can be localized easily when required.
- Rather than localize Authoring templates, you should use the new WCM Text Providers to provide localized template names, element names and help-text all within the one authoring template. Please check section "Text Providers"
- Categories should not be localized but rather WCM Text Providers should be used to provide localized names. Please check section "Text Providers"

Library Copy Portlet

This extension makes it easy to roll out a new locale by copying an existing library, assigning it a new name and locale. This is useful when you mainly create your base content library (*ContentLibraryEN*) with all its structure in terms of site areas, then you copy this library to build a new localized (*ContentLibraryAR*). From this point you can start configuring MLS for this set of related localized libraries.



Follow the steps below to use the extension.

1. Go to the Portal Administration / Portal Content / ML Library Copy page
2. Select a library to copy from the first pull-down
3. Select a new locale from the second pull-down
4. Enter a new name for the library
5. Click the "Copy" button

Our Client MLS Configurations

Assuming you already installed MLS package, or enabled in a version 8 Portal according to the referenced documentation. Before the plugins can be used, the MLS needs to be configured, this is a 7 steps process (please execute the following steps on the main authoring server node only):

1. Modify the security of the *MLConfiguration_v7* library as follows
    a. Create a group (such as "*Locale Owners*") to house all users that will own one the two locales we have (include the Base Locale Owner in this group). The Base Locale is English.
    b. Assign "*Locale Owners*" the "Editor" role on the library.
    c. Assign the "All Portal User Groups" the "Contributor" role on the library
    d. Uncheck the "Allow Inheritance" checkbox for all Item Types except "Content", "Components" (hides all sections in the Authoring UI for the *MLConfiguration_v7* library from "*Locale Owners*" except "Content" and "Components").
    e. Assign the your *SiteAdmins* group to the "Administrator" role in all Item Types and the library.
2. Run the following command from the <PORTAL_PROFILE_DIR> /ConfigEngine directory to fix up the user references within the MLConfiguration_v7 library:
    a. ***Windows:*** *ConfigEngine.bat run-wcm-admin-task-member-fixer -DPortalAdminId=username -DPortalAdminPwd=password -Dlibrary=MLConfiguration_v7 -Dfix=true -DinvalidDn=update -DmismatchedId=update -DaltDn=update*
    b. ***Linux:*** *ConfigEngine.sh run-wcm-admin-task-member-fixer -DPortalAdminId=username -DPortalAdminPwd=password -Dlibrary=MLConfiguration_v7 -Dfix=true -DinvalidDn=update -DmismatchedId=update -DaltDn=update*
3. Configure the authoring portlet so that the *MLConfiguration_v7* library is added to the library selection list.

4. Create the content representing the ML Configuration File for the set of WCM sites you want to manage together
    a. From the WCM Authoring Portlet, navigate to the "*MLConfiguration_v7*‟ library and create a new piece of content using the "*LocalizedConfigurationFileAT*‟ authoring template
    b. Fill in the fields on the Content field as follows:
        i. **Name**: Our Client Portal
        ii. **Display Title**:. Our Client Portal
        iii. **Base Content Library**: ContentLibraryEN
        iv. **Content Libraries**: ContentLibraryEN, ContentLibraryAR
        v. **Has Regionalizations**: false
        vi. **Content Library Owners***: <Place the emails for each locale owner for the mentioned libraries. Each entry in a new line>*

5. In "*ContentLibraryAR*" library, create a WCM Link Component that points back to its ML Configuration file content created in the previous step.
    a. The name of the WCM Link Component is important. It must be "*MLConfFileReference*‟.
    b. The *[ALL_USERS]* group should also be granted the "User‟ role
6. Activate Email Notifications for the workflow synchronization extensions:
    a. From the WCM Authoring Portlet, open an existing ML Configuration file content under the "*MLConfiguration_v7*‟ library
    b. Click the "Edit‟ button
    c. Make the following changes to the "General Workflow Synchronization Settings‟ section:
        i. **emailServer**: *<Provide your email SMTP Service>*
        ii. **emailFromAddress**: *<The email address entered here will be used to set the "From‟ address on all email notifications. This field must be set to a valid email address>*
        iii. **authoringUIURL**: *<This field is used to create links to processed content items. It should be set to the URL of your Authoring server. i.e. http://localhost:10045/wps/myportal/wcmAuthoring>*
        iv. Enable Email notifications for the following extensions:
            1. Set the ***localize.emailNotificationsEnabled*** field in the "Localize Workflow Synchronization Settings‟ section to **true**. This useful to let the

locale owner that there is a content to be translated for the Arabic language.

7. Enable "WCM Project"s integration for synchronized publishing. Set the **SyncPublish.useProjects** field in the "*SyncPublish*" Workflow Synchronization Settings" section to **true**

Note: The *MLConfiguration_v7* library must be syndicated to all staging and production servers like any other library.

Authoring Plugin

The Multilingual authoring plugin is the extension used within the authoring UI to see associated translations of the currently open piece of content AND create new translations if they don't exist.

Follow the steps below to use the extension:

1. To enable the Navigation/Creation links for all your authoring templates, add a text component called "*ML Translations*" to each authoring template, with the following properties

readMode=/wps/wcmml;/jsp/html/MLAuthorTimeRead_AutoLoad.jsp,editMode=/wps/wcmml;/jsp/html/MLAuthorTimeEdit_AutoLoad.jsp

2. Note: Place the new text component above any existing components. It will look better!
3. Save the authoring template and re-apply the template to all content using it ensuring the "add new elements" option is selected
4. Repeat these steps for all your authoring templates

Localize Workflow Stage

Throughout the above authoring plugin, the author may chose to manually copy the current content to mirror it into the other locale i.e. Our client's authors currently saving the English version of an article as a draft, they then choose to "Copy to Arabic" in order for a new draft content copied into the Arabic library at the same site area hierarchy. The newly created content is NOT auto translated, it's copied as is in English, and the author still needs to translate its elements accordingly.

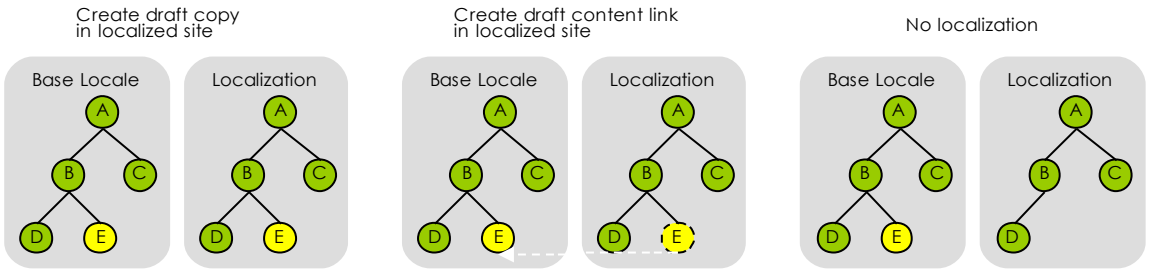
MLS offers a "Localize" workflow stage to automatically create that new content for the author, without the need to manually hit "Copy to Arabic". This happens once the original content reach to this stage. It's up to the designer to place the stage anywhere in the Workflow, it's decided to put it right after the "Draft" workflow stage (See Workflow strategy section)

Once the content gets localized (copied), an email notification sent to the Arabic Library owner letting him know that there is a new draft content copied that needs translation. We have enabled this notification while creating the ML Configuration File content.

The extension can be configured to:

- Copy the content to the new localized library.
- Base content is linked into the new localized library for use non-translated (as-is)
- Nothing happens, the other library is ignored

The default behavior is to copy the content, and this is what is used for our client.

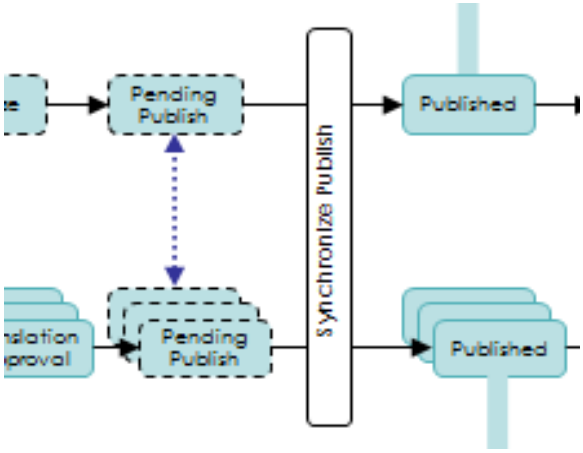


You only need to place Add the "*MLConfiguration_v7 \ Localize*" workflow stage to your workflow (after the Draft stage), in order to get this feature.

Publishing Synchronization Extension

To ensure that the base content and localized contents are published at the same time, we have enabled the publishing synchronization extension by simply setting "*syncPublish.useProjects=true*" in the ML Configuration file.

The idea is that MLS creates a new WCM "Project" once the content gets localized in the other library, the project will not get published until both contents reaches the Publish workflow stage. The Project then deleted.



There is a different legacy approach to enable synchronized publishing mechanism by simply adding a different workflow stage just before the Publish stage. However, when synchronized publishing is required, enabling "WCM Projects" integration (setting syncPublish.useProjects=true in the SyncPublish Workflow Synchronization settings) is strongly recommended as this greatly reduces the load generated by this function.

Other Workflow Synchronization Extensions

MLS offers two more workflow synchronization extensions, namely:

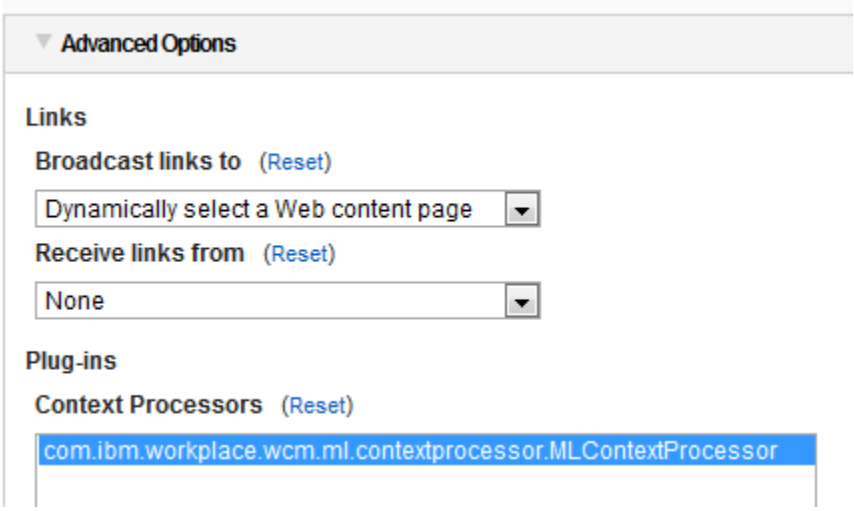
- **Expiry Synchronization:** To ensure that the base item and localized items are expired at the same time, however we are **not** using this extension as we are utilizing the usage of "Schedule Move" action that automatically moves to the contents to the Expiry Stage once the assigned expiry date is due.
- **Delete Synchronization**: To ensure that all localized contents are deleted, if the base content gets deleted, , however we are **not** using this extension as we are utilizing the usage of a custom "Purge" action that permanently deletes to the contents once it reaches to the Deleted workflow stage.

Render-time Navigation Extension

This extension provides the ability to modify the Portal locale and allow the Web Content Viewer Portlets to become locale aware and thus automatically switch to an equivalent object when the locale changes.

Follow the steps below to use the extension:

1. Integrate the "*MLPortalLocaleSwitcher.jsp*" (from /jsp/html of wcm-multilocale.ear) into your Portal theme.
2. Update all WCM Local Rendering Portlets to reference the ML Context Processor: Edit the portlet settings and select the *com.ibm.workspace.wcm.ml.contextProcessor.MLContextProcessor* from within the "Plugins" section of the "Advanced Options".



The Web Content Viewer Portlet will call the configured Context Processor prior to rendering each item. The "*MLContextProcessor*" checks whether the locale has changed. If the locale has changed, then it will see if there is an equivalent item in the new locale and if so, updates the portlet's runtime configuration to tell the portlet to show the localized item.

If the Web Content Viewer Portlet has its presentation template overridden via the configuration, then a localized presentation template is also searched for and used if available.

Locale Switcher JSP

The "MLPortalLocaleSwitcher.jsp" provides the facility to switch the Portal locale by first detecting the current locale and then using the Portal URL API to create links to switch the locale to other available locales as per the list of configured libraries in the ML Configuration file.

The following configuration options are available within the 'MLPortalLocaleSwitcher.jsp':

- **s_displayHeaderAndFooter:** Controls whether the header and footer is displayed. Default is true.
- **s_useCurrentLocaleInLanguageDisplayNames**: Controls whether the available language's are displayed using the current locale (true) or the locale of the language (false). Default is true.

The following query string parameters can be optionally included in a URL to control behavior of an individual request:

- **MLContextProcessorIgnore:** Will tell the context processor to ignore the request.
- **overrideLocale=[locale]:** Will tell the context processor to use the specific locale instead of the current locale when processing the request.

Text Providers

A text provider is used to provide localized text that can be used within web content item forms. For example, a text provider can be used to localize the field labels or help text within an authoring template so that each user sees the labels or help text in their own language.

With Text Providers, we do not need to copy the authoring templates for each locale, instead we provide a centralized source of localized translation for our custom fields.

To use a text provider, you must create a text provider class and then register the text provider by deploying it on the server.

1. In RSA, create New Dynamic Web Project.
2. Prepare a properties file that have ALL the keys/values pairs of all labels to be translated for your client Authoring use cases. Have several properties file, one for each locale.
3. Create a new Java Class which implements *com.ibm.workplace.wcm.api.plugin.textprovider.TextProvider*
4. Implement all methods of *TextProvider* class. Pay attention to *getString(String key, Locale displayLocale)*, where you should return a translated value that corresponds to the passed key according to the passed locale. Also *getProviderKeys(),* which returns all your registered keys.
5. Create new *plugin.xml* under *WEB-INF*
6. Make starting weight of the application as same as starting weight of WCM application; by default it should be 20.
7. Deploy application; a Text Provider will be available in WCM

A sample code of the "OurClientTextProvider" would be:

```
// OurClientTextProvider.java

public class OurClientTextProvider implements TextProvider {

public String getProviderName() {

// This method returns the unique name of the text provider.

return "OurClientTextProvider";

}

public String getString(String key, Locale displayLocale) {

// This method returns some translated text, given a key identifying the message and a locale.

return ResourceBundle.getBundle("your.package.name.wcm.nl.OurClientResourceBundle", displayLocale).getString(key);

}

public Collection<String> getProviderKeys() {

// This method returns a list of keys used when accessing the text provider. These keys are displayed in the authoring UI when // a user is configuring the text provider.
```

```
return ResourceBundle.getBundle("your.package.name.wcm.nl.OurClientResourceBundle", new Locale("en")).keySet();

}

public boolean isShownInAuthoringUI() {

// This method allows you to prevent your text provider from appearing in the authoring UI.

return true;

}

// Methods inherited from com.ibm.portal.Localized must also be implemented

public String getTitle(Locale displayLocale) {

// This method returns the title for the text provider that will be used to allow selection of the text provider.

return "OurClientTextProvider";

}

public ListModel<Locale> getLocales() {

// This method returns a list of locales that are supported by this text provider.

final List<Locale> locales = new ArrayList<Locale>();

locales.add(new Locale("en"));

locales.add(new Locale("ar"));

return new ListModel<Locale>(){Iterator<Locale> iterator(){return locales.listIterator();}};

}

public String getDescription(Locale p_arg0) {

// This method returns a description of the text provider.

return "OurClientTextProvider";

}

}
```

// plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin id="com.ibm.ourclient.workplace.wcm.api.text.provider"
       name="Our Client Text Provider"
       version="1.0.0"
       provider-name="IBM">

 <extension
    point="com.ibm.workplace.wcm.api.TextProvider"
    id="OurClientTextProvider">
    <provider class="your.package.name.wcm.OurClientTextProvider"/>
 </extension>

</plugin>
```

// OurClientResourceBundle_en.properties		// OurClientResourceBundle_ar.properties	
client.article.title =	Title	client.article.title =	العنوان
client.article.picture =	Picture	client.article.picture =	الصورة
client.article.description =	Description	client.article.description =	الوصف
client.video.run =	Play	client.video.run =	تشغيل
client.product.price =	Specify Product Price	client.product.price =	أذكر سعر المنتج

For more details on how to create the Text Provider plugin, check at: Creating a Text Provider class

Related OOTB WCM Functionality

WCM already contains a number of related multilingual features that can be utilized in multilingual sites, these are:

1. Text Providers: This is explained in the previous section
2. Dot notation in Library component references (i.e. <component name="./MyCmpnt">): Allow you to easily localize a library component by copying it into each localized library (with same name) and changing as necessary.
3. 'Current Content' / 'Current SiteArea' options in Menus and Navigators: Allow you to build localized menus and navigators.

Locale-aware URLs

Pages and Labels can be titled according to the selected locale by specifying relevant localized name in their properties. Using the Locale Selector portlets, portal pages/labels pick the appropriate name accordingly. That works well for the page names/titles, however if your requirements demand a localized URLs for these pages, you simply can:

- Implement URL Mapping(s) for the pages (as many URLs as needed for specific page, that matches all locales it covers)
- Set the language for each URL accordingly. Please check this article in InfoCenter (especially the "Setting the language for a URL Mapping").
- The configuration part of this can be a bespoke application (i.e. ANT) that runs in the background to assign languages for created URL in batches.

Even after using the URL mapping mechanism, the URL generated via WCM components or Portal theme would have the long statefull version, that is automatically generated by Portal, so when a user clicks those links, the statefull URL still appears in the address bar. If your client requires a shorter normalized URL, you may:

- Use Servlet Filter to intercept any incoming requests and make a redirect to the short version of the URL defined in the previous mapping.
- Use Portal State pre-processor and perform the same actions done in the filter.

Bilingual Random Text

For a multilingual website, aside from localizing contents, page titles, and labels, in some cases you might need to specify a random text that is not part of any content i.e. "Read More", "Download", "Show Details"...etc. These samples are also required to be localized.

First thing to do is to extract these instances and put each in a specified HTML Component, for it to be used across the site. We are not localizing WCM components and we have a single design library used for both languages, English and Arabic. In case of "Learn More", for example, it will be rendered as such in case English locale is selected, but it should be "إقرأ المزيد" if Arabic locale is selected.

To overcome this issue we will create a commonly used HTML Component that utilizes the use of WCM Rendering Plugins, to show the appropriate text/link of the current content, based on the current preferred locale.

The *LearnMore* HTML Component might look like the below snippet that used Matches and Locale plugin together. Use this ONLY as a reference.

```
[Plugin:Matches pattern="en(.*)" text="[Plugin:Locale]"]
```

```
<a href="[placeholder tag="href"] ">Learn More</a>

[/Plugin:Matches]

[Plugin:Matches pattern="ar(.*)" text="[Plugin:Locale]"]

<a href="[placeholder tag="href"] ">إقرأ المزيد </a>

 [/Plugin:Matches]
```

Similarly The *Details* HTML Component might look like the below snippet that used Matches and Locale plugin together. Use this ONLY as a reference

```
[Plugin:Matches pattern="en(.*)" text="[Plugin:Locale]"]

<a href="[placeholder tag="href"] ">Details</a>

[/Plugin:Matches]

[Plugin:Matches pattern="ar(.*)" text="[Plugin:Locale]"]

<a href="[placeholder tag="href"] ">التفاصيل </a>

 [/Plugin:Matches]
```

Conclusion

Multilingual requirements have been mandatory in, noticeably, in every extranet website and in most of intranet sites. A successful Web Experience Management solution is the one that provides a solid out-of-the-box framework and approach to deliver such requirements with minimal customization and no limitations.

Here you have learned how to effectively make a use of IBM WCM MLS solution together with other features related to localization within the product in order to fulfill your client needs.