XMLUI Configuration and Customization

The DSpace digital repository supports two user interfaces: one based on JavaServer Pages (JSP) technologies and one based upon the Apache Cocoon framework (XMLUI). This chapter describes those parameters which are specific to the Manakin (XMLUI) interface based upon the Cocoon framework.

1 Overview of XMLUI / Manakin
    1.1 Understanding the Flow of an XMLUI Request
2 Manakin Configuration Property Keys
3 Configuring Themes and Aspects
    3.1 Aspects
    3.2 Themes
4 Multilingual Support
5 MathJax Support
6 Creating a New Theme
7 Customizing the News Document
8 Adding Static Content
9 Harvesting Items from XMLUI via OAI-ORE or OAI-PMH
    9.1 Automatic Harvesting (Scheduler)
10 Additional XMLUI Learning Resources

Overview of XMLUI / Manakin

For more information & diagrams

For a more detailed overview of XMLUI/Manakin, see the following resources:

- Introducing Manakin (XMLUI) - Provides an overview of what XMLUI is and how it works.
- Making DSpace Your Own (DSpace 6.x edition) - Recorded webinar which walks through basic tasks of modifying XMLUI in DSpace 6.x
- Learning to Use Manakin (XMLUI) - Overview of how to use Manakin and how it works. Based on DSpace 1.5, but also valid for current versions
- Making DSpace XMLUI Your Own - Concentrates on using Maven to build Overlays in the XMLUI (Manakin). Also has very basic examples for JSPUI. Based on DSpace 1.6.x but also valid for current versions.
- Manakin theme tutorial - with good debugging tips

The XMLUI (aka Manakin) is built on Apache Cocoon framework. The XMLUI uses Cocoon to provide a modular, extendable, tiered interface framework

The XMLUI essentially consists of three main tiers, in increasing order of complexity:

1. **Style Tier** - allows one to use CSS and simple XHTML to stylize an existing XMLUI Theme
2. **Theme Tier** - allows one to use XSLT, XHTML and CSS to create new, more complex XMLUI Theme(s)
3. **Aspect Tier** - allows one to use the Cocoon framework and Java (or XSLT) to create new features (aspects), and generate new content into DRI.

These tiers are very important and powerful because of their modularity. For example, based on your local expertise with these technologies, your institution may decide to only modify the XMLUI at the "Style Tier" (by just modifying CSS & images in an existing theme). As you learn more about themes & aspects, you may decide to slowly venture into the more complex "Theme Tier" and finally into the "Aspect Tier". Other institutions may determine that all they really need to ever do is make "Style Tier" changes.

Digging in a little deeper, there are three main XMLUI components that are unique to the XMLUI and used throughout the system. These main components are:

- DRI Schema- Digital Repository Interface (DRI) XML schema, which is the "abstract representation of a single repository page". The DRI document is XML that contains all of the information (metadata) available for display on a given page within the XMLUI. This information includes:
    - Metadata elements (described in METS, MODS, DSpace Internal Metadata (DIM), Qualified Dublin Core, etc.)
    - Structural elements (described in TEI light)
    - For more specific information about DRI Schema along with examples, see DRI Schema Reference.
- #Aspects - One or more aspects are enabled at a given time. Generally speaking an aspect implements a set of related features within the XMLUI. More specifically, the enabled aspects are what **build** the DRI document. So, Aspects are the only things that can change the structure of the DRI document (or add/remove content to/from DRI)
    - Aspects apply to all pages across your entire DSpace site. Each aspect must take a valid DRI document as its input, and also output a valid DRI document.
    - Aspects usually are written in Java (and controlled by a Cocoon "sitemap.xmap"). However, Aspects can also be written in XSLT (provided that the input and output are both valid DRI documents)
- #Themes- One or more themes are enabled at a given time. Themes are in charge of stylizing content into a particular look & feel. More specifically, a theme is what transforms a DRI document into XHTML (and adds any CSS, javascript, images, etc).
    - A single Theme may apply to your entire DSpace site, just a specific Community or Collection (and all members of that Community /Collection), or just a specific page.
    - A Theme may consist of one or more of the following: an XSLT stylesheet, CSS stylesheets, images, other static resources.
    - More information on creating a theme is available at: #Creating a New Theme
    - Additional Theme Resources include:
        - Manakin theme tutorial
        - Manakin Themes and Recipes
        - Create a new theme (Manakin)

Understanding the Flow of an XMLUI Request

One of the harder things to initially grasp in the XMLUI is how a single user's request (e.g. clicking on a link or button) flows through the entire system of enabled Aspects and Themes. Understanding this flow is also very important as you work to build your own Aspects (or complex Themes), as it may allow you to more easily determine what is going on in the system.

Before getting started, it's worth mentioning that this request flow is controlled via a series of Cocoon Sitemap files (named sitemap.xmap, themes.xmap and aspects.xmap). These Sitemap files are Cocoon's way of defining the flow. More information about Cocoon Sitemaps is available at: http://cocoon.apache.org/2.1/userdocs/concepts/sitemap.html

The following explanation provides a high level overview of how a request is processed, how a DRI document is generated (via Aspects), and then how it is transformed into XHTML (via Themes). As this is a high level overview, some details are likely left out, but the overarching flow is what is most important.

1. A user visits an XMLUI page (by clicking a link or button, etc)
2. That request begins in the root Cocoon `sitemap.xmap` (located at `[xmlui]/sitemap.xmap`). This is the main entry point for **all requests**
    a. Within that sitemap, various URL path matching takes place. If the request is to download a document, that document is returned immediately.
    b. However, in many cases, the request is for a page within the XMLUI. In this scenario, the root sitemap.xmap will load the `[xmlui]/themes/themes.xmap`file, which controls all the Themes.
        i. The `themes.xmap` file will then load all "matching" themes which are configured in your `[dspace]/config/xmlui.xconf` file (see #Themes below).
        ii. If more than one theme matches the current URL path, then the **first match wins**
        iii. Once a matching theme is located, that theme's `sitemap.xmap`file (located in its theme directory) is loaded and processed.
            1. The theme's `sitemap.xmap`is in charge of actually loading the theme's XSLT, CSS, etc. However, before it does that, you'll notice it makes a call to generate the DRI document for the current page as follows:

```
<map:generate type="file" src="cocoon://DRI/{1}"/>
```

            2. This DRI call generates a brand new, internal Cocoon request. This request is then processed back in the **root** `sitemap.xmap` (remember how we said that this sitemap is the main entry point for all requests).
3. Back in the root sitemap, the "DRI/**" call is matched. This causes the `[xmlui]/aspects/aspects.xmap`file to be loaded. As the name suggests, this file obviously controls all the Aspects.
    a. The `aspects.xmap` file will then load all enabled Aspects which are configured in your `[dspace]/config/xmlui.xconf` file (see #Aspects below).
    b. Each aspect is loaded in the **order that it appears**. However, **multiple** aspects may be loaded for the same URL path. *Remember, aspects can build upon each other (we call this an "aspect chain") as they work together to generate the final DRI document.*
    c. When an Aspect is loaded, its `sitemap.xmap` is loaded & processed
        - NOTE: An aspect's sitemap.xmap is actually compiled into the `dspace-xmlui-api.jar` file. However, if you have a copy of DSpace source handy, it can be found in: `[dspace-src]/dspace-xmlui/dspace-xmlui-api/src/main/resources/aspects/[name-of-aspect]/`
    d. Each aspect is processed one-by-one (again in the order they are listed in `xmlui.xconf`). Each aspect may add, remove or change content within the DRI document. After the final aspect is finished processing, the DRI document is complete.
        - HINT: In the XMLUI you can always view the final DRI document by adding "?XML" or "&XML" on to the end of the current URL in your web browser. See more instructions for debugging in Manakin theme tutorial.
4. Once the final DRI document is complete (all aspects are done processing), the flow will return back to your Theme's `sitemap.xmap` (remember, this is the same location that triggered the loading of the Aspects in the first place).
5. At this point, your Theme's `sitemap.xmap` will continue its processing. Generally speaking, most themes will then perform one or more XSLT transformations (to transform the final DRI document into XHTML). They also may load up one or more CSS files to help stylize the final XHTML.
6. Finally, once the Theme has completed its processing (remember, only one theme is ever processed for a single request), the final generated XHTML document is displayed to the user.

Again, the above flow is a slightly simplified version of what is going on underneath the XMLUI. As you can see, Cocoon Sitemaps are what control most of the XMLUI processing (and the loading of the Aspects and Theme).

Manakin Configuration Property Keys

In an effort to save the programmer/administrator some time, the configuration table below is taken from 5.3.43. *XMLUI Specific Configuration*.

Property:	*xmlui.supportedLocales*
Example Value:	*xmlui.supportedLocales = en, de*
Informational Note:	A list of supported locales for Manakin. Manakin will look at a user's browser configuration for the first language that appears in this list to make available to in the interface. This parameter is a comma separated list of Locales. All types of Locales country, country_language, country_language_variant. Note that if the appropriate files are not present (i.e. Messages_XX_XX.xml) then Manakin will fall back through to a more general language.

Property:	*xmlui.force.ssl*
Example Value:	*xmlui.force.ssl = true*
Informational Note:	Force all authenticated connections to use SSL, only non-authenticated connections are allowed over plain http. If set to true, then you need to ensure that the '*dspace.hostname*' parameter is set to the correctly.
Property:	*xmlui.user.registration*
Example Value:	*xmlui.user.registration = true*
Informational Note:	Determine if new users should be allowed to register. This parameter is useful in conjunction with Shibboleth where you want to disallow registration because Shibboleth will automatically register the user. Default value is true.
Property:	*xmlui.user.editmetadata*
Example Value:	*xmlui.user.editmetadata = true*
Informational Note:	Determines if users should be able to edit their own metadata. This parameter is useful in conjunction with Shibboleth where you want to disable the user's ability to edit their metadata because it came from Shibboleth. Default value is true.
Property:	*webui.user.assumelogin*
Example Value:	*webui.user.assumelogin = true*
Informational Note:	Determine if super administrators (those who are in the Administrator group) can login as another user from the "edit eperson" page. This is useful for debugging problems in a running dspace instance, especially in the workflow process. The default value is false, i.e., no one may assume the login of another user.
Property:	*xmlui.user.loginredirect*

Example Value:	*xmlui.user.loginredirect = /profile*
Informational Note:	After a user has logged into the system, which url should they be directed? Leave this parameter blank or undefined to direct users to the homepage, or */profile* for the user's profile, or another reasonable choice is */submissions* to see if the user has any tasks awaiting their attention. The default is the repository home page.
Property:	*xmlui.theme.allowoverrides*
Example Value:	*xmlui.theme.allowoverrides = false*
Informational Note:	Allow the user to override which theme is used to display a particular page. When submitting a request add the HTTP parameter "themepath" which corresponds to a particular theme, that specified theme will be used instead of the any other configured theme. Note that this is a potential security hole allowing execution of unintended code on the server, this option is only for development and debugging it should be turned off for any production repository. The default value unless otherwise specified is "false".
Property:	*xmlui.bundle.upload*
Example Value:	*xmlui.bundle.upload = ORIGINAL, METADATA, THUMBNAIL, LICENSE, CC_LICENSE*
Informational Note:	Determine which bundles administrators and collection administrators may upload into an existing item through the administrative interface. If the user does not have the appropriate privileges (add and write) on the bundle then that bundle will not be shown to the user as an option.
Property:	*xmlui.community-list.render.full*
Example Value:	*xmlui.community-list.render.full = true*
Informational Note:	On the community-list page should all the metadata about a community/collection be available to the theme. This parameter defaults to true, but if you are experiencing performance problems on the community-list page you should experiment with turning this option off.
Property:	*xmlui.community-list.cache*
Example Value:	*xmlui.community-list.cache = 12 hours*

Informational Note:	Normally, Manakin will fully verify any cache pages before using a cache copy. This means that when the community-list page is viewed the database is queried for each community/collection to see if their metadata has been modified. This can be expensive for repositories with a large community tree. To help solve this problem you can set the cache to be assumed valued for a specific set of time. The downside of this is that new or editing communities/collections may not show up the website for a period of time.
Property:	*xmlui.bistream.mods*
Example Value:	*xmlui.bistream.mods = true*
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The MODS metadata file must be inside the "METADATA" bundle and named MODS.xml. If this option is set to 'true' and the bitstream is present then it is made available to the theme for display.
Property:	*xmlui.bitstream.mets*
Example Value:	*xmlui.bitstream.mets = true*
Informational Note:	Optionally, you may configure Manakin to take advantage of metadata stored as a bitstream. The METS metadata file must be inside the "METADATA" bundle and named METS.xml. If this option is set to "true" and the bitstream is present then it is made available to the theme for display.
Property:	*xmlui.google.analytics.key*
Example Value:	*xmlui.google.analytics.key = UA-XXXXXX-X*
Informational Note:	If you would like to use google analytics to track general website statistics then use the following parameter to provide your analytics key. First sign up for an account at http://analytics.google.com, then create an entry for your repositories website. Google Analytics will give you a snipit of javascript code to place on your site, inside that snip it is your Google Analytics key usually found in the line: _uacct = "UA-XXXXXX-X" Take this key (just the UA-XXXXXX-X part) and place it here in this parameter.
Property:	*xmlui.controlpanel.activity.max*
Example Value:	*xmlui.controlpanel.activity.max = 250*
Informational Note:	Assign how many page views will be recorded and displayed in the control panel's activity viewer. The activity tab allows an administrator to debug problems in a running DSpace by understanding who and how their DSpace is currently being used. The default value is 250.

Property:	*xmlui.controlpanel.activity.ipheader*
Example Value:	*xmlui.controlpanel.activity.ipheader = X-Forward-For*
Informational Note:	Determine where the control panel's activity viewer receives an events IP address from. If your DSpace is in a load balanced environment or otherwise behind a context-switch then you will need to set the parameter to the HTTP parameter that records the original IP address.

Configuring Themes and Aspects

The Manakin user interface is composed of two distinct components: *aspects* and *themes*. Manakin aspects are like extensions or plugins for Manakin; they are interactive components that modify existing features or provide new features for the digital repository. Manakin themes stylize the look-and-feel of the repository, community, or collection.

The repository administrator is able to define which aspects and themes are installed for the particular repository by editing the *[dspace]/config/xmlui.xconf* configuration file. The *xmlui.xconf* file consists of two major sections: Aspects and Themes.

Aspects

The <*aspects*> section defines the "Aspect Chain", or the linear set of aspects that are installed in the repository. For each aspect that is installed in the repository, the aspect makes available new features to the interface. For example, if the "submission" aspect were to be commented out or removed from the *xmlui.xconf*, then users would not be able to submit new items into the repository (even the links and language prompting users to submit items are removed). Each <*aspect*> element has two attributes, *name* and *path*. The name is used to identify the Aspect, while the path determines the directory where the aspect's code is located. Here is the default aspect configuration:

```
<aspects>
    <aspect name="Displaying Artifacts" path="resource://aspects/ViewArtifacts/" />
    <aspect name="Browsing Artifacts" path="resource://aspects/BrowseArtifacts/" />
    <aspect name="Searching Artifacts" path="resource://aspects/SearchArtifacts/" />
    <aspect name="Administration" path="resource://aspects/Administrative/" />
    <aspect name="E-Person" path="resource://aspects/EPerson/" />
    <aspect name="Submission and Workflow" path="resource://aspects/Submission/" />
    <aspect name="Statistics" path="resource://aspects/Statistics/" />
    <aspect name="Original Workflow" path="resource://aspects/Workflow/" />
</aspects>
```

A standard distribution of Manakin/DSpace includes eight "core" aspects:

- **ViewArtifacts** The ViewArtifacts Aspect is reposonsible for displaying individual item metadata.
- **BrowseArtifacts** The BrowseArtifacts Aspect is reponsible for displaying different browse options
- **SearchArtifacts** The SearchArtifacts Aspect displays the different search boxes. **Shouldn't be activated together with DSpace Discovery**.
- **Administrative** The Administrative Aspect is responsible for administrating DSpace, such as creating, modifying and removing all communities, collections, e-persons, groups, registries and authorizations.
- **E-Person** The E-Person Aspect is responsible for logging in, logging out, registering new users, dealing with forgotten passwords, editing profiles and changing passwords.
- **Submission** The Submission Aspect is responsible for submitting new items to DSpace, determining the workflow process and ingesting the new items into the DSpace repository.
- **Statistics** The Statistics Aspect is responsible for displaying statistics information.
- **Workflow** The Original Workflow Aspect is responsible for displaying workflow tasks. **Shouldn't be activated with the new workflow called XMLWorkflow**

Following Aspects are optional

- **XMLWorkfflow** This Aspect was added in DSpace 1.8 and is responsible for the new configurable workflow system. **Shouldn't be activated together with the Workflow aspect**.
- **Discovery** The Discovery Aspect replaces the standard search with faceted searching. It also takes care of the faceted browse options. **Shouldn't be activated togather with SearchArtifacts**.
- **SwordClient** The SwordClient aspect displays options that allow you to "push" DSpace content to another SWORD-server enabled system.
- **XMLTest** An aspect to assist developers in creating themes, as it displays different debugging options.

Following Aspects are deprecated and shouldn't be used anymore at all

- **ArtifactBrowser** This aspect has been split up into ViewArtifacts, BrowseArtifacts and SearchArtifacts in DSpace 1.7.0

Themes

The <*themes*> section defines a set of "rules" that determine where themes are installed in the repository. Each rule is processed in the order that it appears, and the first rule that matches determines the theme that is applied (so order is important). Each rule consists of a <*theme*> element with several possible attributes:

- **name** (*always required*)The name attribute is used to document the theme's name.
- **path** (*always required*)The path attribute determines where the theme is located relative to the *themes/* directory and must either contain a trailing slash or point directly to the theme's *sitemap.xmap* file.
- **regex** (*either regex and/or handle is required*)The regex attribute determines which URLs the theme should apply to.
- **handle** (*either regex and/or handle is required*)The handle attribute determines which community, collection, or item the theme should apply to. If you use the "handle" attribute, the effect is cascading, meaning if a rule is established for a community then all collections and items within that community will also have this theme apply to them as well. Here is an example configuration:

```
<themes>
    <theme name="Theme 1" handle="123456789/23" path="theme1/"/>
    <theme name="Theme 2" regex="community-list"        path="theme2/"/>
    <theme name="Reference Theme" regex=".*" path="Reference/"/>
</themes>
```

In the example above three themes are configured: "Theme 1", "Theme 2", and the "Reference Theme". The first rule specifies that "Theme 1" will apply to all communities, collections, or items that are contained under the parent community "123456789/23". The next rule specifies any URL containing the string "community-list" will get "Theme 2". The final rule, using the regular expression "**.", will match *anything**, so all pages which have not matched one of the preceding rules will be matched to the Reference Theme.

Multilingual Support

The XMLUI user interface supports multiple languages through the use of internationalization catalogues as defined by the Cocoon Internationalization Transformer. Each catalog contains the translation of all user-displayed strings into a particular language or variant. Each catalog is a single xml file whose name is based upon the language it is designated for, thus:

messages_*language_country_variant*.xml

messages_*language_country*.xml

messages_*language*.xml

messages.xml

The interface will automatically determine which file to select based upon the user's browser and system configuration. For example, if the user's browser is set to Australian English then first the system will check if *messages_en_au.xml* is available. If this translation is not available it will fall back to *messages_en.xml*, and finally if that is not available, *messages.xml*.

DSpace XMLUI supplies an English only translation of the interface, which can be found in the XMLUI web application (`[dspace]/webapps/xmlui/i18n/messages.xml`), after you first build DSpace.

If you wish to add other translations to the system, or make customizations to the existing messages.xml file, you can place them in the following directory:

```
[dspace-source]/dspace/modules/xmlui/src/main/webapp/i18n/
```

After rebuilding DSpace, any messages files placed in this directory will be automatically included in the XMLUI web application (and files of the same name will override any default files). By default this full directory path may not exist (if not, just create it) or may be empty. You can place any number of translation catalogues in this directory. To add additional translations, just add alternative versions of the *messages.xml* file in specific language and country variants as needed for your installation.

To set a language other than English as the default language for the repository's interface, you can simply rename the translation catalogue for the new default language to "*messages.xml*".

Again, note that you will need to rebuild DSpace for these changes to take effect in your installed XMLUI web application!

For more information about the `[dspace-source]/dspace/modules/` directory, and how it may be used to "overlay" (or customize) the default XMLUI interface, classes and files, please see: Advanced Customisation

MathJax Support

The Mirage and Mirage 2 themes of the XMLUI user interface support rendering of math formulae via the inclusion of the MathJax JavaScript library. This feature was added to DSpace 5 with DS-635, and from version 5 through version 6.2 a single dollar sign (`$...$`) was the configured delimiter for math formulae. With DS-3087, the default delimiters (`$$...$$` and `\[...\]` for displayed mathematics, and `\(...\)` for in-line mathematics) of MathJax are used–this is true for DSpace versions 6.3 and higher. For more help with using MathJax, see the MathJax documentation.

Creating a New Theme

Manakin themes stylize the look-and-feel of the repository, community, or collection and are distributed as self-contained packages. A Manakin/DSpace installation may have multiple themes installed and available to be used in different parts of the repository. The central component of a theme is the sitemap.xmap, which defines what resources are available to the theme such as XSL stylesheets, CSS stylesheets, images, or multimedia files.

1) Create theme skeleton

Most theme developers do not create a new theme from scratch; instead they start from the standard theme template, which defines a skeleton structure for a theme. The template is located at: *[dspace-source]/dspace-xmlui/dspace-xmlui-webbapp/src/main/webbapp/themes/template*. To start your new theme simply copy the theme template into your locally defined modules directory, *[dspace-source]/dspace/modules/xmlui/src/main/webbapp/themes/[your theme's directory]/*.

2) Modify theme variables

The next step is to modify the theme's parameters so that the theme knows where it is located. Open the *[your theme's directory]/sitemap.xmap* and look for *<global-variables>*

```
<global-variables>
    <theme-path>[your theme's         directory]</theme-path>
    <theme-name>[your theme's name]</theme-name>
</global-variables>
```

Update both the theme's path to the directory name you created in step one. The theme's name is used only for documentation.

3) Add your CSS stylesheets

The base theme template will produce a repository interface without any style - just plain XHTML with no color or formatting. To make your theme useful you will need to supply a CSS Stylesheet that creates your desired look-and-feel. Add your new CSS stylesheets:

[your theme's directory]/lib/style.css (The base style sheet used for all browsers)

[your theme's directory]/lib/style-ie.css (Specific stylesheet used for internet explorer)

4) Install theme and rebuild DSpace

Next rebuild and deploy DSpace (replace <version> with the your current release):

1. Rebuild the DSpace installation package by running the following command from your *[dspace-source]/dspace*/directory:

   ```
   mvn package
   ```

2. Update all DSpace webapps to *[dspace]/webapps* by running the following command from your *[dspace-source]/dspace/target/dspace-[version]-build.dir*directory:

   ```
   ant -Dconfig=[dspace]/config/dspace.cfg update
   ```

3. Deploy the the new webapps:

   ```
   cp -R /[dspace]/webapps/* /[tomcat]/webapps
   ```

4. Restart Tomcat
   This will ensure the theme has been installed as described in the previous section "Configuring Themes and Aspects".

Customizing the News Document

The XMLUI "news" document is only shown on the root page of your repository. It was intended to provide the title and introductory message, but you may use it for anything.

The news document is located at *[dspace]/dspace/config/news-xmlui.xml*. There is only one version; it is localized by inserting "i18n" callouts into the text areas. It must be a complete and valid XML DRI document (see Chapter 15).

Its (the News document) exact rendering in the XHTML UI depends, of course, on the theme. The default content is designed to operate with the reference themes, so when you modify it, be sure to preserve the tag structure and e.g. the exact attributes of the first DIV tag. Also note that the text is DRI, not HTML, so you must use only DRI tags, such as the XREF tag to construct a link.

Example 1: a single language:

```
<document>
    <body>
        <div id="file.news.div.news" n="news" rend="primary">
            <head> TITLE OF YOUR REPOSITORY HERE </head>
            <p>
                INTRO MESSAGE HERE
                Welcome to my wonderful repository etc etc ...
                A service of <xref target="http://myuni.edu/">My University</xref>
            </p>
        </div>
    </body>
    <options/>
    <meta>
        <userMeta/>
        <pageMeta/>
        <repositoryMeta/>
    </meta>
</document>
```

Example 2: all text replaced by references to localizable message keys:

```
<document>
    <body>
        <div id="file.news.div.news" n="news" rend="primary">
            <head><i18n:text>myuni.repo.title</i18n:text></head>
            <p>
                <i18n:text>myuni.repo.intro</i18n:text>
                <i18n:text>myuni.repo.a.service.of</i18n:text>
                <xref target="http://myuni.edu/"><i18n:text>myuni.name</i18n:text></xref>
            </p>
        </div>
    </body>
    <options/>
    <meta>
        <userMeta/>
        <pageMeta/>
        <repositoryMeta/>
    </meta>
</document>
```

Adding Static Content

The XMLUI user interface supports the addition of globally static content (as well as static content within individual themes).

Globally static content can be placed in the *[dspace-source]/dspace/modules/xmlui/src/main/webapp/static/* directory. By default this directory only contains the default *robots.txt* file, which provides helpful site information to web spiders/crawlers. However, you may also add static HTML (**.html*) content to this directory, as needed for your installation.

Any static HTML content you add to this directory may also reference static content (e.g. CSS, Javascript, Images, etc.) from the same *[dspace-source] /dspace/modules/xmlui/src/main/webapp/static/* directory. You may reference other static content from your static HTML files similar to the following:

```
<link href="./static/mystyle.css" rel="stylesheet" type="text/css"/>
<img src="./static/images/static-image.gif" alt="Static image in /static/images/ directory"/>
<img src="./static/static-image.jpg" alt="Static image in /static/ directory"/>
```

Harvesting Items from XMLUI via OAI-ORE or OAI-PMH

This feature allows you to harvest Items (both metadata and bitstreams) from one DSpace to another DSpace or from one OAI-PMH/OAI-ORE server to a DSpace instance.

This section will give the necessary steps to set up the OAI-ORE/OAI-PMH Harvester from the XMLUI (Manakin). This feature is currently not available in the JSPUI.

Setting up a Harvesting Collection:

1. Login to XMLUI and create a new collection.

2. Go to the tab named "Content Source" that appears next to "Edit Metadata" and "Assign Roles " in the collection edit screens.
3. The two "Content Source" options are "standard DSpace collection" (selected by default) and "collection harvests its content from an external source". Select "harvests from an external source" option and click Save.
4. A new set of menus appear to configure the harvesting settings:
    - "OAI Provider" is in the URL of the OAI-PMH provider that the content from this collection should be harvested from. The OAI-PMH provider deployed with DSpace typically has the format: `http://dspace.url/oai/request` For example, you could use the Demo DSpace OAI-PMH provider: "http://demo.dspace.org/oai/request"
    - "OAI Set Id" is the OAI-PMH setSpec of the collection you wish to harvest from. For DSpace, this Set ID has the format: `hdl_<handle-prefix>_<handle-suffix>`. For example "hdl_10673_2" would refer to the Collection whose handle is "10673/2" (on the DSpace Demo Server, this is the Collection of Sample Items). If the target instance is using OAI 2.0 (DSpace 3.0 or the OAI 2.0 addon for DSpace 1.8.2), replace "hdl_" with "col_" if the set is a collection or with "com_" if it's a community.
    - "Metadata format" determines the format that the descriptive metadata will be harvested. The OAI-PMH server of the source DSpace instance may only support certain metadata formats. Select "DSpace Intermediate Metadata" if available (as this provides the richest metadata transfer) and "Simple Dublin Core" otherwise
        - To determine which metadata formats an OAI-PMH server supports, you can send a `ListMetadataFormats` request to that OAI-PMH server. Typically this has the format: `http://dspace.url/oai/request?verb=ListMetadataFormats` For example, you can see which metadata formats are supported by the DSpace Demo Server by visiting: http://demo.dspace.org/oai/request?verb=ListMetadataFormats
    - Click the "Test Settings" button to verify the settings supplied in the previous steps. This will usually let you know if anything is missing or does not validate correctly. If you receive an error, you will need to fix the settings before continuing
5. The list of radio buttons labeled "Content being harvested" allows you to select the level of harvest. These harvesting options include:
    - *Harvest Metadata Only* - will only harvest item metadata from the source DSpace (or any OAI-PMH source)
    - *Harvest metadata and references to bitstreams (requires ORE support)* - will harvest item metadata and create links to files/bitstreams (stored remotely) from the source DSpace (requires OAI-ORE)
    - *Harvest metadata and bitstreams (requires ORE support)* - performs a full local replication. Harvests both item metadata and files /bitstreams (requires OAI-ORE).
6. Select the appropriate option based on your needs, and click Save

At this point the settings are saved and the menu changes to provide three options:

- *Change Settings* : takes you back to the edit screen (see above instructions)
- *Import Now*: performs a single harvest from the remote collection into the local one. Success, notes, and errors encountered in the process will be reflected in the "Last Harvest Result" entry. More detailed information is available in the DSpace log.

  "Import Now" May Timeout for Large Harvests

  Note that the whole harvest cycle is executed within a single HTTP request and will time out for large collections. For this reason, it is advisable to use the automatic harvest scheduler set up either in XMLUI or from the command line. If the scheduler is running, "Import Now" will handle the harvest task as a separate thread.
- *Reset and Reimport Collection* : will perform the same function as "Import Now", but will clear the collection of all existing items before doing so.

Automatic Harvesting (Scheduler)

Setting up automatic harvesting in the Control Panel Screen.

- Login as an Administrative user in XMLUI
- Visit the "Harvesting" tab under "Administrative > Control Panel"
- The panel offers the following information:
    - Available actions:
        - *Start Harvester* : starts the scheduler. From this point on, all properly configured collections (listed on the next line) will be harvested at regular intervals. This interval can be changed in the `dspace.cfg` using the `harvester.harvestFrequency` parameter.
        - *Pause* : the "nice" stop; waits for the active harvests to finish, saves the state/progress and pauses execution. Can be either resumed or stopped.
        - *Stop* : the "full stop"; waits for the current item to finish harvesting, and aborts further execution.
        - *Reset Harvest Status* : since stopping in the middle of a harvest is likely to result in collections getting "stuck" in the queue, the button is available to clear all states.

Additional XMLUI Learning Resources

Useful links with further information into XMLUI Development

- Making DSpace XMLUI Your Own - Concentrates on using Maven to build Overlays in the XMLUI (Manakin). Also has very basic examples for JSPUI. Based on DSpace 1.6.x.
- Learning to Use Manakin (XMLUI) - Overview of how to use Manakin and how it works. Based on DSpace 1.5, but also valid for 1.6.
- Introducing Manakin (XMLUI)