Enhanced Configuration Scheme

Enhanced Configuration Scheme was accepted into the DSpace 6.0 release

The Enhanced Configuration Scheme will first be released in DSpace 6.0. This documentation is in the process of being merged/moved over into the official configuration documentation for DSpace 6.
This page describes the enhanced/reloadable configuration feature, based on Apache Commons Configuration, which was added in DSpace 6.

- Ticket: Unable to locate Jira server for this macro. It may be due to Application Link configuration.
- PR: http

TESTERS NEEDED! While the basics of this functionality work (see PR above), this change literally changes how **every** configuration is read by DSpace (as Apache Commons Configuration has its own enhanced Property file syntax, see below for more on that).

This means it's likely that some specific features (especially optional ones) may need to have their configuration file/settings tweaked. I've done my best to already fix the configurations of out-of-the-box features, but have not yet tested all optional features.

Overview

In DSpace 5 or below, DSpace used its own custom Property-based configuration scheme, along with a custom `build.properties` which could tweak the build/compilation process in order to "override" some pre-selected configurations in the `dspace.cfg` file. While this configuration scheme "worked" at a basic level, it required a lot of custom variable interpolation (i.e. filtering) to occur in both the Maven build process (`mvn package`) and the Ant install process (`ant fresh_install` or `ant update`). The end result was that configuration files in your DSpace installation directory (`[dspace.dir]`) contained the correct settings with all variables (`${setting}`) replaced by the values from your build.properties file. So, it was no longer possible to easily tweak certain key settings (like `dspace.dir` or `solr.server`) without having to either re-run the entire build process or make corrections to several files at once.

Enter Apache Commons Configuration.

Since DSpace 6, the Enhanced Configuration Scheme feature uses Apache Commons Configuration (version 1.10) as the new configuration scheme for DSpace. This provides several key advantages over our old, custom configuration scheme:

- Apache Commons Configuration is a well-established Java library whose goal is to make configuration more flexible and easier to manage.
- It automatically interpolates all settings at *runtime*. This means we no longer need to replace variables (`${setting}`) within our configurations. They will be auto-determined at runtime based on the value of that variable within one of the configuration files. For more on variable interpolation see its Basic Features documentation.
- It is a flexible configuration scheme. It can read configurations from several sources at once, including Properties files, XML config files and even database tables (see its Overview documentation). Currently, in the DSpace Enhanced Configuration Scheme we are still only using Properties files, similar to DSpace 5 and below. But, should we decide to do so, we are now able to easily move all or some configurations to XML configs or database config tables.
  - The locations of the configuration sources can be easily customized by DSpace administrators in a new `config-definition.xml` file, which configures Apache Commons Configuration for DSpace. More on that below.
  - The `config-definition.xml` file itself is simply a "configuration definition" file as defined by Apache Commons Configuration. See the Configuration File Documentation for more details.
- It allows for easy overriding of configuration values from other sources. How the overrides occur depends on how you've configured Apache Commons Configuration. For DSpace, we have a new `config-definition.xml` which defines the following override scheme (again, this can be easily tweaked for local needs):
  - If a setting is specified in Java System Properties (e.g. `-D[setting]=[value]`), it overrides the same setting found in any below location
  - If a setting is specified as an Environment Variable, it overrides the same setting found in any below location
  - If a setting is specified in the new `local.cfg` configuration file, it overrides the default value in any below location
  - Default values for all settings are specified in the `dspace.cfg` and the `modules/*.cfg` configuration files.
- It supports *enhanced* Properties files. This means our `dspace.cfg` , `local.cfg` and other configuration files can now immediately support some enhanced options, including:
  - The ability to easily **include** other configuration files via: `"include=[config-file-location]"` (See the end of the updated `dspace.cfg` for examples)
  - The ability to provide lists of values to "array" configurations by specifying the setting multiple times (rather than creating a giant comma separated configuration spanning multiple lines). For example, enabling *both* LDAP and Password authentication can now be done via these two lines:

        - `plugin.sequence.org.dspace.authenticate.AuthenticationMethod = org.dspace.authenticate.LDAPAuthentication`
        - `plugin.sequence.org.dspace.authenticate.AuthenticationMethod = org.dspace.authenticate.PasswordAuthentication`
    - For more information see the Commons Config Properties File documentation.
- More information/features can also be found in the Apache Commons Configuration v1.10 User Guide.

Building / Installing DSpace

With the Enhanced Configuration Scheme, the DSpace build process is slightly changed. The `build.properties` file no longer exists and therefore has no effect on the build process.

Here's the basics of building/installing DSpace:

- Download DSpace (as usual)
- `cd [dspace-source]`
- Create your own initial local.cfg configuration file
    - `cp local.cfg.EXAMPLE local.cfg`
- The following fields MUST be specified in your local.cfg in order to install DSpace:
    - `dspace.dir`
    - database connection information (`db.url`, `db.driver`, `db.dialect`, `db.username`, `db.password`, `db.schema`)
    - All other fields are optional, and can be specified at a later time, or not at all. (As you'll read later on in these instructions, any configuration can also be *added* to your local.cfg).
- Build/Compile/Install as normal
    - `mvn clean package`
    - `ant fresh_install` (or `ant update`)
- Once DSpace is installed, your local.cfg will be copied over to your `[dspace.dir]/config/` location. At that time you can optionally tweak it further (see local.cfg documentation below)

Unlike the old `build.properties`, the new `local.cfg` has NO effect on the Maven build process.

It is ONLY used by Ant to determine the location where DSpace should be installed/updated (using `dspace.dir`), and also to initialize/update the database (using `db.*` settings).
Many configuration names/keys have changed!

If you are upgrading from an earlier version of DSpace, you will need to be aware that *many* configuration names/keys have changed. Because Apache Commons Configuration allows for auto-overriding of configurations, all configuration names/keys in different `*.cfg` files MUST be uniquely named (otherwise accidental, unintended overriding may occur).

In order to compensate for this, all `modules/*.cfg` files had their configurations **renamed** to be prepended with the module name. As a basic example, all the configuration settings within the `modules/oai.cfg` configuration now start with `"oai."`.

Additionally, while the `local.cfg` may look *similar* to the old `build.properties`, many of its configurations have slightly different names. So, simply copying your build.properties into a local.cfg will NOT work.

This means that DSpace 5.x (or below) configurations are NOT compatible with the Enhanced Configuration Scheme. While you obviously can use your old configurations as a reference, you will need to start with fresh copy of all configuration files, and reapply any necessary configuration changes (this has always been the recommended procedure). However, as you'll see in the next section, you'll likely want to do that anyways in order to take full advantage of the new `local.cfg` file.

local.cfg

The `[dspace.dir]/config/local.cfg` file is the new way to customize your DSpace configuration based on your local needs.

There are a few key things to note about this configuration file:

- Any setting in your `local.cfg` will automatically OVERRIDE a setting of the same name in the `dspace.cfg` or any `modules/*.cfg` file. This also means that you can copy ANY configuration (from `dspace.cfg` or any `modules/*.cfg` file) into your `local.cfg` to specify a new value.
    - For example, specifying `dspace.url` in `local.cfg` will override the default value of `dspace.url` in `dspace.cfg`.
    - Also, specifying `oai.solr.url` in `local.cfg` will override the default value of `oai.solr.url` in `config/modules/oai.cfg`
- The `local.cfg` file is an Apache Commons Configuration Property file. For more information see the Commons Config Properties File documentation
    - This means it has enhanced features like the ability to include other config files (via `"include="` statements).
- As needed, you also are able to OVERRIDE settings in your `local.cfg` by specifying them as System Properties or Environment Variables.
    - For example, if you wanted to change your `dspace.dir` in development/staging environment, you could specify it as a System Property (e.g. `-Ddspace.dir=[new-location]`). This new value will override any value in *both* `local.cfg` and `dspace.cfg`.

An example local.cfg is provided at `[dspace-source]`/local.cfg.EXAMPLE. The example only provides a few key configurations which all DSpace sites are likely to need to customize. However, you may add (or remove) any other configuration to your `local.cfg` to customize it as you see fit.

Link to local.cfg.EXAMPLE: https://github.com/DSpace/DSpace/blob/master/local.cfg.EXAMPLE

config-definition.xml

Link to config-definition.xml: https://github.com/DSpace/DSpace/blob/master/dspace/config/config-definition.xml

The `[dspace.dir]/config/config-definition.xml` file defines the Apache Commons Configuration settings that DSpace utilizes by default. It is a valid "configuration definition" file as defined by Apache Commons Configuration. See the Configuration File Documentation for more details.

You are welcome to customize the `config-definition.xml` to customize your local configuration scheme as you see fit. Any customizations to this file will require restarting your servlet container (e.g. Tomcat).

By default, the DSpace `config-definition.xml` file defines the following configuration:

- All DSpace configurations are loaded via Properties files
  - Note: Apache Commons Configuration does support other configuration sources such as XML configurations or database configurations (see its Overview documentation).
- *Configuration Files/Sources:* By default, only two configuration files are loaded into Apache Commons Configuration:
  - `local.cfg` (see documentation on local.cfg above.)
  - `dspace.cfg` (NOTE: however that all `modules/*.cfg` are loaded by `dspace.cfg` via "`include=`" statements at the end of that configuration file.)
- *Configuration Override Scheme:* The configuration override scheme is defined as follows. Configurations specified in earlier locations will automatically override any later values:
  - System Properties (-D[setting]=[value]) override all other options
  - Environment Variables
  - `local.cfg`
  - `dspace.cfg` (and all `modules/*.cfg` files) contain the default values for all settings.
- *Configuration Auto-Reload:* By default, all configuration files are automatically checked each minute for changes. If they have changed, they are automatically reloaded.

Configuration Reloading and Caching

As noted above, by default, DSpace will now automatically reload any modified configuration file (`local.cfg`, `dspace.cfg` or `modules/*.cfg`) within one minute.

While the new values are immediately available within the DSpace ConfigurationService, some configurations may still be "cached" within UI-specific code. This often occurs when a UI (or API) loads a configuration value into a `static` variable, or otherwise implements/provides its own object caching mechanism.

The Enhanced Configuration Scheme codebase does NOT attempt to correct all these instances of caching within UIs or APIs. This would require individual configurations to be tested and any caching mechanisms to be removed.

FAQs

Can I have different local.cfg files for different environments (e.g. development/testing/staging /production)?

Yes, but you'll need to tweak the default configuration scheme. By default, DSpace does NOT allow you to have multiple `local.cfg` files (one per environment). However, with some minimal tweaks to your configuration scheme, you *likely* (untested) could achieve this in one of two ways:

1. Change your `config-definition.xml` to use a system property (of your choice) instead of the hardcoded name "local.cfg". The Configuration Definition file itself does allow for variables to be included, but they must be specified in a previous configuration source (in that config-definition.xml) or via a system property. See the Configuration File Documentation for more details. So, you could simply change your config-definition.xml to use a "`dspace.env`" system property, and pass "`-Ddspace.env=dev`" to have it use a `[dspace.dir]/config/dev.cfg`:

   ```
   <!-- Change local.cfg to be ${dspace.env} in your config-definition.xml -->
   <properties fileName="${dspace.env}.cfg" throwExceptionOnMissing="false" config-name="local" config-
   optional="true">
   ...
   </properties>

   <!-- OPTIONALLY: If you wanted to have some default local configs shared among *all* environments, you
   could add
        a NEW "properties" file to always load those defaults. In this example, default.cfg would be loaded
   for ALL
        environments. Configs in the environment-specific ${dspace.env}.cfg would override default.cfg, and
        both would override dspace.cfg (and other *.cfg). -->
   <properties fileName="default.cfg" throwExceptionOnMissing="false" config-name="default" config-
   optional="true">
   ...
   </properties>
   ```

2. Alternatively, you could use the "`include=`" option (of Apache Commons Configuration Properties Files) within your `local.cfg` file to load a *different* configuration file, again based on a setting specified as a system property. For example, your `local.cfg` file would ONLY consist of "`include=`" statement(s), which would load whichever configuration file was specified as the "dspace.env" system property:

```
# This is the ENTIRE local.cfg -- all settings would instead be located in environment-specific config
files.
# Its job is just to load up the configuration for the environment specified by "dspace.env"
# For example, -Ddspace.env=dev would load [dspace.dir]/config/dev.cfg
#         and  -Ddspace.env=prod would load [dspace.dir]/config/prod.cfg

# Load the environment-specific file
include = ${dspace.env}.cfg

# OPTIONALLY: If you wanted to have some default local configs shared among *all* environments, you
could add
# a second "include=" statement to always load those defaults from a file of your choice. In this
example,
# a default.cfg would be loaded for ALL environments. Configs in the environment-specific ${dspace.env}.
cfg
# would override default.cfg, and both would override dspace.cfg (and other *.cfg).
include = default.cfg
```

While the above examples both use a property named `${dspace.env}`, you can use whatever property you want. The name itself doesn't matter. Additionally, both show examples of using a "`default.cfg`" to specify properties which are shared between several environments. This file can also be named whatever you want. Just tweak the name(s) in the examples above to meet your local needs.

The option you choose above would likely depend on your own local practices/needs. Either of these options should work, provided that you place your environment-specific configuration files within the `[dspace.dir]/config` directory alongside the `local.cfg` file.

Advanced Topics

Configuration Interpolation

This is less important to normal users of DSpace, but may be of high interest to developers and some system administrators.

Configuration variables determined at runtime

It's important to be aware of the fact that variables within the following types of configurations are now AUTOMATICALLY interpolated at *runtime* using Apache Commons Configuration (and our ConfigurationService). This means that variables (`${setting}`) are no longer filtered by Maven or Ant for any of the following configuration types. In other words, variables are perfectly OK in these configuration files in your DSpace installation directory (i.e. `[dspace`).

- Primary Configuration files (namely `local.cfg`, `dspace.cfg` and all `modules/*.cfg`)
- Primary Log4j settings (`log4j.properties`)
- Spring XML configs (namely `[dspace.dir]/config/spring/api/*.xml`)

Configuration variables filtered during installation (prior to runtime)

There are a few configuration file(s) which still require their variables/settings to be filtered/interpolated during installation. The following configuration files are still filtered during the Installation/Update process (`ant fresh_install` or `ant update`), and *cannot be determined at runtime*. In other words, variables *cannot* exist in these configuration files in your DSpace installation directory (i.e. `[dspace]`).

- `web.xml` files still require filtering, both to support IDE integration, and to properly initialize all webapps in your Servlet Container (e.g. Tomcat).
  - To support IDE integration (and allow debugging of webapps from IDEs), all `web.xml` files are filtered by Maven using the `filteringDeploymentDescriptors` setting in POMs. Without this setting, the web.xml files will never be filtered when attempting to run any DSpace webapp from within an IDE.
  - Additionally, to support running the webapps in general, the `${dspace.dir}` variable is also filtered (by Ant). This is because the `dspace.dir` context parameter in these `web.xml` files is used to initialize the DSpace Kernel (and tell the webapp where the DSpace home directory is). Unfortunately, there's no way to interpolate this value at runtime as the `dspace.dir` value does not exist until the Kernel and the ConfigurationService have initialized.
    - In other words, the DSpace webapps cannot function/initialize without a `dspace.dir`. We either need to filter a value for it (during ant update/fresh_install), or we need to REQUIRE that it be specified by other means.
    - The only way we'd get around this problem would be to REQUIRE a `dspace.dir` ALWAYS be specified to the servlet container (as a Context parameter and/or system property).
- `robots.txt` : Obviously there's no way for a static file like robots.txt to load configurations at runtime. This file is filtered by Ant during a "fresh_install" or "update".
- `log4j-*.properties` : While the primary `log4j.properties` configuration is NOT filtered, DSpace also includes several other log4j files which are utilized by third-party dependencies (e.g. Solr uses its own log4j-solr.properties). As these third-party dependencies have their own initialization process, they cannot utilize DSpace's ConfigurationService, and their log4j configurations must be filtered by Ant during a "fresh_install" or "update".
- RDF configurations: The DSpace RDF / Linked Data interface has TTL configuration files which require minor filtering. These files are filtered by Ant during a "fresh_install" or "update".
- OAI-PMH `description.xml`: This static, custom OAI-PMH configuration file requires minor filtering. This file is filtered by Ant during a "fresh_install" or "update".

Java API Changes

ConfigurationManager vs ConfigurationService

In the DSpace 5 Java API, we had two types of Configuration objects: `org.dspace.coreConfigurationManager` and `org.dspace.services.ConfigurationService`.

While the the `ConfigurationManager` still exists in the API (and is still called by some areas of the codebase), it is now a "wrapper" object. It simply wraps calls to the configured `ConfigurationService`.

As before, the default ConfigurationService is the `org.dspace.servicemanager.config.DSpaceConfigurationService` (in dspace-services).

The `DSpaceConfigurationService` has been updated/enhanced to utilize Apache Commons Configuration, and to better align its methods with the old `ConfigurationManager` class. It also has added a new `reloadConfig()` method which can be called on demand to automatically reload all configurations.

PluginManager vs PluginService

In DSpace 5, the `org.dspace.core.PluginManager` class managed all DSpace "plugin" definitions (i.e. `plugin.*` settings in `dspace.cfg`). (SIDENOTE: these DSpace "plugin" definitions are simply Java interfaces, which are then mapped to classes which implement that plugin interface).

While this concept still exists (and all plugin configurations are still respected/valid), the `PluginManager` itself has been entirely replaced by a new `org.dspace.core.service.PluginService.` This change was necessary in order to "Spring-ify" the `PluginManager` and make it compatible with the `ConfigurationService`. In prior releases (5.x and below), the `PluginManager` was highly dependent on the `ConfigurationManager`, and as such, did not respect/follow the Spring bean initialization process. In other words, without this major refactor, the PluginManager would attempt to request configurations from the ConfigurationService *before* the ConfigurationService was fully initialized by Spring.

The default PluginService is a new `org.dspace.core.LegacyPluginServiceImpl` class, which implements the functionality of the old `PluginManager`.