Transforming DSpace Content (MediaFilters)

MediaFilters: Transforming DSpace Content

Overview

DSpace can apply filters or transformations to files/bitstreams, creating new content. Filters are included that extract text for **full-text searching**, and create **thumbnails** for items that contain images. The media filters are controlled by the `dspace filter-media` script which traverses the asset store, invoking all configured `MediaFilter` or `FormatFilter` classes on files/bitstreams (see Configuring Media Filters for more information on how they are configured).

Available Media Filters

Below is a listing of all currently available Media Filters, and what they actually do:

Name	Java Class	Function	Enabled by Default?
HTML Text Extractor	`org.dspace.app.mediafilter.HTMLFilter`	extracts the full text of HTML documents for full text indexing. (Uses Swing's HTML Parser)	true
JPEG Thumbnail	`org.dspace.app.mediafilter.JPEGFilter`	creates thumbnail images of GIF, JPEG and PNG files	true
Branded Preview JPEG	`org.dspace.app.mediafilter.BrandedPreviewJPEGFilter`	creates a branded preview image for GIF, JPEG and PNG files	false
PDF Text Extractor	`org.dspace.app.mediafilter.PDFFilter`	extracts the full text of Adobe PDF documents (only if text-based or OCRed) for full text indexing. (Uses the Apache PDFBox tool)	true
XPDF Text Extractor	`org.dspace.app.mediafilter.XPDF2Text`	extracts the full text of Adobe PDF documents (only if text-based or OCRed) for full text indexing (Uses the XPDF command line tools available for Unix.) See XPDF Filter Configuration for details on installing/enabling.	false
Word Text Extractor	`org.dspace.app.mediafilter.WordFilter`	extracts the full text of Microsoft Word or Plain Text documents for full text indexing. (Uses the "Microsoft Word Text Mining" tools.)	true
PowerPoint Text Extractor	`org.dspace.app.mediafilter.PowerPointFilter`	extracts the full text of slides and notes in Microsoft PowerPoint and PowerPoint XML documents for full text indexing (Uses the Apache POI tools.)	true

Please note that the `filter-media` script will automatically update the DSpace search index by default (see ReIndexing Content (for Browse or Search)) This is the recommended way to run these scripts. But, should you wish to disable it, you can pass the **-n** flag to either script to do so (see Executing (via Command Line) below).

Enabling/Disabling MediaFilters

The media filter plugin configuration `filter.plugins` in `dspace.cfg` contains a list of all enabled media/format filter plugins (see Configuring Media Filters for more information). By modifying the value of `filter.plugins` you can disable or enable MediaFilter plugins.

Executing (via Command Line)

The media filter system is intended to be run from the command line (or regularly as a cron task):

```
[dspace]/bin/dspace filter-media
```

With no options, this traverses the asset store, applying media filters to bitstreams, and skipping bitstreams that have already been filtered.

Available Command-Line Options:

- **Help** : `[dspace]/bin/dspace filter-media -h`
  - Display help message describing all command-line options.
- **Force mode** : `[dspace]/bin/dspace filter-media -f`
  - Apply filters to ALL bitstreams, even if they've already been filtered. If they've already been filtered, the previously filtered content is overwritten.

- **Identifier mode** : `[dspace]/bin/dspace filter-media -i 123456789/2`
  - Restrict processing to the community, collection, or item named by the identifier - by default, all bitstreams of all items in the repository are processed. The identifier must be a Handle, not a DB key. This option may be combined with any other option.
- **Maximum mode** : `[dspace]/bin/dspace filter-media -m 1000`
  - Suspend operation after the specified maximum number of items have been processed - by default, no limit exists. This option may be combined with any other option.
- **No-Index mode** : `[dspace]/bin/dspace filter-media -n`
  - Suppress index creation - by default, a new search index is created for full-text searching. This option suppresses index creation if you intend to run `index-update` elsewhere.
- **Plugin mode** : `[dspace]/bin/dspace filter-media -p "PDF Text Extractor","Word Text Extractor"`
  - Apply ONLY the filter plugin(s) listed (separated by commas). By default all named filters listed in the *filter.plugins* field of *dspace.cfg* are applied. This option may be combined with any other option. *WARNING:* multiple plugin names must be separated by a comma (i.e. ',') and NOT a comma followed by a space (i.e. ', ').
- **Skip mode** : `[dspace]/bin/dspace filter-media -s 123456789/9,123456789/100`
  - SKIP the listed identifiers (separated by commas) during processing. The identifiers must be Handles (not DB Keys). They may refer to items, collections or communities which should be skipped. This option may be combined with any other option. *WARNING:* multiple identifiers must be separated by a comma (i.e. ',') and NOT a comma followed by a space (i.e. ', ').
  - NOTE: If you have a large number of identifiers to skip, you may maintain this comma-separated list within a separate file (e.g. *filter-skiplist.txt*). Use the following format to call the program. *Please note the use of the "grave" or "tick" (`) symbol and do not use the single quotation.*
    - `[dspace]/bin/dspace filter-media -s `less filter-skiplist.txt``
- **Verbose mode** : `[dspace]/bin/dspace filter-media -v`
  - Verbose mode - print all extracted text and other filter details to STDOUT.
    Adding your own filters is done by creating a class which *implements* the `org.dspace.app.mediafilter.FormatFilter` interface. See the Creating a new Media/Format Filter topic and comments in the source file `FormatFilter.java` for more information. In theory filters could be implemented in any programming language (C, Perl, etc.) However, they need to be invoked by the Java code in the Media Filter class that you create.

Creating Custom MediaFilters

Creating a simple Media Filter

New Media Filters **must implement** the *org.dspace.app.mediafilter.FormatFilter* interface. More information on the methods you need to implement is provided in the *FormatFilter.java* source file. For example:

```
public class MySimpleMediaFilter implements FormatFilter
```

Alternatively, you could extend the *org.dspace.app.mediafilter.MediaFilter* class, which just defaults to performing no pre/post-processing of bitstreams before or after filtering.

```
public class MySimpleMediaFilter extends MediaFilter
```

You must give your new filter a "name", by adding it and its name to the *plugin.named.org.dspace.app.mediafilter.FormatFilter* field in *dspace.cfg*. In addition to naming your filter, make sure to specify its input formats in the *filter.<class path>.inputFormats* config item. Note the input formats must match the *short description* field in the Bitstream Format Registry (i.e. *bitstreamformatregistry* table).

```
plugin.named.org.dspace.app.mediafilter.FormatFilter = \
        org.dspace.app.mediafilter.MySimpleMediaFilter = My Simple Text Filter, \ ...

filter.org.dspace.app.mediafilter.MySimpleMediaFilter.inputFormats =
        Text
```

If you neglect to define the *inputFormats* for a particular filter, the *MediaFilterManager* will never call that filter, since it will never find a bitstream which has a format matching that filter's input format(s).

If you have a complex Media Filter class, which actually performs different filtering for different formats (e.g. conversion from Word to PDF **and** conversion from Excel to CSV), you should define this as described in Chapter 13.3.2.2 .

Creating a Dynamic or "Self-Named" Format Filter

If you have a more complex Media/Format Filter, which actually performs **multiple** filtering or conversions for different formats (e.g. conversion from Word to PDF **and** conversion from Excel to CSV), you should have define a class which implements the *FormatFilter* interface, while also extending the Chapter 13.3.2.2 *SelfNamedPlugin* class. For example:

```
public class MyComplexMediaFilter extends SelfNamedPlugin implements FormatFilter
```

Since *SelfNamedPlugins* are self-named (as stated), they must provide the various names the plugin uses by defining a getPluginNames() method. Generally speaking, each "name" the plugin uses should correspond to a different type of filter it implements (e.g. "Word2PDF" and "Excel2CSV" are two good names for a complex media filter which performs both Word to PDF and Excel to CSV conversions).

Self-Named Media/Format Filters are also configured differently in *dspace.cfg*. Below is a general template for a Self Named Filter (defined by an imaginary *MyComplexMediaFilter* class, which can perform both Word to PDF and Excel to CSV conversions):

```
#Add to a list of all Self Named filters
plugin.selfnamed.org.dspace.app.mediafilter.FormatFilter = \
        org.dspace.app.mediafilter.MyComplexMediaFilter
#Define input formats for each "named" plugin this filter implements
        filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Word2PDF.inputFormats = Microsoft Word
        filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Excel2CSV.inputFormats = Microsoft Excel
```

As shown above, each Self-Named Filter class must be listed in the `plugin.selfnamed.org.dspace.app.mediafilter.FormatFilter` item in `dspace.cfg`. In addition, each Self-Named Filter **must** define the input formats for *each named plugin* defined by that filter. In the above example the *MyComplexMediaFilter* class is assumed to have defined two named plugins, `Word2PDF` and `Excel2CSV`. So, these two valid plugin names ("Word2PDF" and "Excel2CSV") **must** be returned by the `getPluginNames()` method of the `MyComplexMediaFilter` class.

These named plugins take different input formats as defined above (see the corresponding *inputFormats* setting).

If you neglect to define the `inputFormats` for a particular named plugin, the `MediaFilterManager` will never call that plugin, since it will never find a bitstream which has a format matching that plugin's input format(s).

For a particular Self-Named Filter, you are also welcome to define additional configuration settings in *dspace.cfg*. To continue with our current example, each of our imaginary plugins actually results in a different output format (Word2PDF creates "Adobe PDF", while Excel2CSV creates "Comma Separated Values"). To allow this complex Media Filter to be even more configurable (especially across institutions, with potential different "Bitstream Format Registries"), you may wish to allow for the output format to be customizable for each named plugin. For example:

```
#Define output formats for each named plugin
filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Word2PDF.output Format = Adobe PDF
filter.org.dspace.app.mediafilter.MyComplexMediaFilter.Excel2CSV.outputFormat = Comma Separated Values
```

Any custom configuration fields in *dspace.cfg* defined by your filter are ignored by the *MediaFilterManager*, so it is up to your custom media filter class to read those configurations and apply them as necessary. For example, you could use the following sample Java code in your *MyComplexMediaFilter* class to read these custom *outputFormat* configurations from *dspace.cfg*:

```
#Get "outputFormat" configuration from dspace.cfg
String outputFormat =  ConfigurationManager.getProperty(MediaFilterManager.FILTER_PREFIX + "." +
MyComplexMediaFilter.class.getName() + "." + this.getPluginInstanceName() + ".outputFormat");
```