Configuration Manual

This is a manual for configuration of the software in a distributed environment. It requires some technical background to understand and use this manual.

This manual describes how to configure the NetarchiveSuite web archive software package. It includes a description of how configurations are set and how to configure use of plugins including how to set up different kinds of repositories.

The deploy software offers a way to make configurations gathered in a special configuration file, which ease the job of configuration. The Installation Manual includes a manual for use of the deploy module to set-up settings for a full distributed system via a configuration file. Using the deploy module will ease the configuration, installation and start/stop of the entire system.

Contents

The first part describes basics of configuration, how it works etc. The second part describes configurations of various items e.g. plug-ins, notifications. The third part introduces special deploy settings which works with the deploy module (referring to Installation Manual).

Note that use of the deploy module (see the Installation Manual) can ease the configuration and installation of !NetarchiveSuite considerably.

This manual does not explain how to install the system (see the Installation Manual for this), extend the functionality of the system (see the development project), or how to use the running system (see the User Manual for this).

- Configuration Basics - NetarchiveSuite Settings
- Detailed Configurations
- Deploy Configurations
- Heritrix Configurations
- Wayback Configuration
- Configuring External Software
- BatchGUI
- Appendix A - Plug-ins in NetarchiveSuite
- Appendix B - Managing Heritrix Harvest Templates (order.xml)
- Appendix C - Migrate the Heritrix templates to NetarchiveSuite 3.6.0+

Audience

The intended audience of this manual is system administrators who will be responsible for the actual setup of !NetarchiveSuite as well as technical personnel responsible for the proper operation of NetarchiveSuite. Some familiarity with XML and Java is an advantage in understanding this manual.

Search manual

Download as pdf\

Basics

Configuration Basics - NetarchiveSuite Settings

Contents

- Setting basics
- Setting keys with multiple values
- Default Settings
    - Common part
    - Harvester part
    - Archive part
    - Viewerproxy (Access) part
    - Monitor part

It is possible to control much of the behaviour of NetarchiveSuite tools and applications using settings. Some settings need to be updated for a distributed system to work, others work best with their default settings.

Below, the basics of settings and default settings are described. For description of how to tailor the configurations to the applications, please refer to the Installation Manual.

Setting basics

All NetarchiveSuite applications are based on the same type of configuration: Keys can be mapped to values, and the mappings can be set either in a settings file written in XML, or on the command line. If no value is specified for a given configuration key, a default value is used.

The keys are defined in a hierarchy. When naming the keys, we separate the levels in a key with dots, for instance:

```
settings.common.http.port=8076
```

When describing the same keys in XML, we use the XML hierarchy:

```
<settings>
  <common>
    <http>
      <port>8076</port>
    </http>
  </common>
</settings>
```

Setting keys with multiple values

Some settings allow a list of values, rather than just one value. For instance:

```
<settings>
  <archive>
    <bitarchive>
      <baseFileDir>/mnt/storage1</baseFileDir>
      <baseFileDir>/mnt/storage2</baseFileDir>
    </bitarchive>
  </archive>
</settings>
```

It is only possible to specify multiple values using configuration files. This cannot be done on the command line.

If you specify more than one settings file, the first settings file to contain a value for the key specifies **all** values. Values from the settings files will not be merged.

As an example, consider the following two settings files:

settings1:

```
<settings>
  <archive>
    <bitarchive>
      <baseFileDir>/mnt/storage1</baseFileDir>
      <baseFileDir>/mnt/storage2</baseFileDir>
    </bitarchive>
  </archive>
</settings>
```

settings2:

```
<settings>
  <archive>
    <bitarchive>
      <baseFileDir>/mnt/storage3</baseFileDir>
      <baseFileDir>/mnt/storage4</baseFileDir>
    </bitarchive>
  </archive>
</settings>
```

The following command will give the value

```
/mnt/storage5
```

:

```
    java -Ddk.netarkivet.settings.file=settings1.xml:settings2.xml
-Dsettings.archive.bitarchive.baseFileDir=/mnt/storage5
dk.netarkivet.common.webinterface.GUIApplication
```

The following command will give the values

```
/mnt/storage1
```

and

```
/mnt/storage2
```

:

```
    java -Ddk.netarkivet.settings.file=settings1.xml:settings2.xml
dk.netarkivet.common.webinterface.GUIApplication
```

The following command will give the values

```
/mnt/storage3
```

and

```
/mnt/storage4
```

:

```
    java -Ddk.netarkivet.settings.file=settings2.xml:settings1.xml
dk.netarkivet.common.webinterface.GUIApplication
```

Default Settings

The NetarchiveSuite package includes such XML setting files with default values for the settings that are used to initialize classes if they are not overwritten by separate settings files or on the command line (please refer to Installation Manual).

The NetarchiveSuite has five main levels under the top settings level:

- common
- harvester
- archive
- viewerproxy
- monitor
  All settings are defined within these five main levels.

The NetarchiveSuite package includes default values for most defined settings. These are defined in XML setting files that are used to initialize classes, one for each main level and one for each plug-in. (TODO: Name the exceptions)

The meaning of the different settings are documented in the javadoc of the associated setting classes as listed below.

Common part

In the common part of the settings, we have general purpose settings (e.g. settings.common.tmpDir, settings.common.http.port), and settings, that allow us to select plug-ins and their associated arguments (e.g. settings.common.!RemoteFile.class, settings.common.jms.broker, settings.common.arcrepositoryClient, and settings.common.indexClient.class).

Most default values for the common part can be found in dk/netarkivet/common/settings.xml and their documentation can be found in the javadoc of the related dk.netarkivet.common.CommonSettings.java class definition.

Futhermore, there are other dedicated common default values for specific plug-in classes defined in the following setting files. All of these are referred to as part of the common part, but are defined with the plug-in itself. Please see section Plug-in Default Settings.

Harvester part

In the harvester part of the settings, we have settings configuring the harvesting process: scheduling, job splitting etc. Most of these settings are used by the scheduler in DefinitionsSiteSection of the GUIApplication

The default values for the harvester part can be found in dk/netarkivet/harvester/settings.xml and their documentation can be found in javadoc of the associated dk.netarkivet.harvester.HarvesterSettings.java class definition.

Archive part

In the archive part of the settings, we have settings related to archive-access (e.g. certain timeouts, replicas and their credentials are defined here). Also behaviour of the !BitarchiveApplications is set here.

The default values for the archive part can be found in dk/netarkivet/archive/settings.xml and their documentation can be found in javadoc of the associated dk.netarkivet.archive.ArchiveSettings.java class definition.

Viewerproxy (Access) part

In the viewerproxy part of the settings, we have settings related to the user-access viewerproxy module (e.g. the main directory used for storing the Lucene index for the jobs being viewed)

The default values for the viewerproxy part can be found in dk/netarkivet/viewerproxy/settings.xml and their documentation can be found in javadoc of the associated dk.netarkivet.viewerproxy.ViewerProxySettings.java class definition.

Monitor part

In the monitor part of the settings, we have settings for the monitoring shown in the System State in the form of e.g. JMX user name and password and number of shown logged lines.

The default values for the monitor part can be found in dk/netarkivet/monitor/settings.xml and their documentation can be found in javadoc of the associated dk.netarkivet.monitor.MonitorSettings.java class definition.

Plug-in default settings

At the moment, the following plugins have associated default settings defined in the following classes, where their documentation can be found in the javadoc:

- EMailNotifications.java with defaults in dk.netarkivet.common.utils.EMailNotificationsSettings.xml.
- FTPRemoteFileSettings.xml with defaults in https://sbforge.org/svn/netarchivesuite/trunk/src/dk/netarkivet/common/distribute/FTPRemoteFileSettings.xml.
- HTTPRemoteFile.java with defaults in dk.netarkivet.common.distribute.HTTPSRemoteFileSettings.xml.
- HTTPSRemoteFile.java with defaults in https://sbforge.org/svn/netarchivesuite/trunk/src/dk/netarkivet/common/distribute/HTTPSRemoteFileSettings.xml
- JMSConnectionSunMQ.java with defaults in dk.netarkivet.common.distribute.JMSConnectionSunMQSettings.xml
- JMSArcRepositoryClient.java with defaults in dk.netarkivet.archive.arcrepository.distribute.JMSArcRepositoryClientSettings.xml
- IndexRequestClient.java with defaults in dk.netarkivet.archive.indexserver.distribute.IndexRequestClientSettings.xml

Detailed Configurations

Contents

Configure Channel Names

Channels are used for communication between applications. There are defined a set of different channel names based on the following settings:

- `settings.common.environmentName`
. This setting is used as prefix to all channel names created in a NetarchiveSuite installation, and must be the same for all applications in the same installation. Note that this means that several installations can be installed on the same machines as long as their environment name is different (e.g. for test installations). The value for the environmentName setting must not contain the character '_'.
- `settings.common.applicationInstanceId`
. This setting is used to distinguish channels when there are more than one of the same application running on the same machine, e.g. when more harvesters are running on the same machine or more bitarchive applications are running on the same machine. Note that also tools like RunBatch and Upload need a distinctt application Instance Id in order to avoid channel name clashes with other applications when communicating with the bitarchives.
- `settings.common.useReplicaId`
. This setting is used to choose the channels for a specific bitarchive in a distributed archive installation. The Replica Id specified must match one of the bitarchive replicas in the `settings.common.replicas` settings. Note that if there is only one bitarchive (or a simple repository installation on local disc) the default values will be sufficient.

Note that some channel names also include the IP address of the machine where the application is running. This is not part of the settings, but ensures that applications on different machines do not share channels when they are not meant to.

For further information, see JMS Channels.

Configure Plug-ins

Parts of the NetarchiveSuite code allow plugging in your own Java implementation, or selecting between different implementations provided by NetarchiveSuite.

When this is done it has two implications on settings:

1. You need to set the implementing class with a setting (these settings always end in `.class`
2. The plug-in may specify extra settings for that plug-in

For list of different plug-ins in the NetarchiveSuite package please refer to Appendix A - Plug-ins in NetarchiveSuite.

For more details on how to extend the system with pluggable classes with their own settings, please see the System Design on plugins.

An example of a plug-in with extra settings is the setting for HTTPRemoteFile.java (extending AbstractRemoteFile.java) which is defined in HTTPRemoteFileSettings.xml:

```
<settings>
    <common>
        <remoteFile>
            <port>8100</port>
        </remoteFile>
    </common>
</settings>
```

Configure Notifications

NetarchiveSuite can send notifications of serious system warnings or failures to the system-owner by email. This is implemented using the Notifications plug-in (see also Appendix A - Plug-ins in NetarchiveSuite). Several settings in the settings.xml can be changed for this to work:

The setting settings.common.notifications.receiver (recipient of notifications), settings.common.notifications.sender (the official sender of the email, and receiver of any bounces), and settings.common.mail.server (the proper mail-server to use):

```
<settings>
  <common>
    <notifications>
        <!-- Which class to instantiate to handle error notifications -->
        <class>dk.netarkivet.common.utils.EmailNotifications</class>
        <!-- The receiver of emails -->
        <receiver>example@netarkivet.dk</receiver>
        <!-- The stated sender of emails  (and receiver of bounces)-->
        <sender>example@netarkivet.dk</sender>
    </notifications>
    <!-- Settings for sending email. Currently mail is only used for email notifications. -->
    <mail>
      <!-- The email server to use -->
      <server>examplesmtpserver.netarkivet.dk</server>
    </mail>
  </common>
</settings>
```

Alternatively, the class dk.netarkivet.common.utils.PrintNotifications can be used. This will simply print the notifications to stderr on the terminal.

```
<settings>
  <common>
    <notifications>
       <!-- Which class to instantiate to handle error notifications -->
       <class>dk.netarkivet.common.utils.PrintNotifications</class>
    </notifications>
  </common>
</settings>
```

Configure a File Data Transfer Method

The data transfer method can be configured as a plug-in (see also Appendix A - Plug-ins in NetarchiveSuite).

You can currently choose between FTP, HTTP, or HTTPS as the filetransfer method. The HTTP transfer method uses only a single copy per transfer, while the FTP method first copies the file to an FTP server and then copies it from there to the receiving side. Additionally, the HTTP transfer method reverts to simple filesystem copying whenever possible to optimize transfer speeds. However, to use HTTP transfers you must have ports open into most machines, which some may consider a security risk. The HTTPS transfer method meets this problem by having the HTTP communication secured and encrypted. To use the HTTPS transfer method you will need to generate a certificate that is needed to contact the embedded HTTPS server.

The FTP method requires one or more FTP-servers installed. (See Installing and configuring FTP for further details). The XML below is an example settings.xml, in which you have to replace serverName, userName, userPassword with proper values. This must be set for all applications to use FTP remote files.

```
<settings>
  <common>
    <remoteFile>
      <!-- The class to use for RemoteFile objects. -->
      <class>dk.netarkivet.common.distribute.FTPRemoteFile</class>
      <!-- The default FTP-server used -->
      <serverName>hostname</serverName>
      <!-- The FTP-server port used -->
      <serverPort>21</serverPort>
      <!-- The FTP username -->
      <userName>exampleusername</userName>
      <!-- The FTP password -->
      <userPassword>examplepassword</userPassword>
      <!-- The number of times FTPRemoteFile should try before giving up
           a copyTo operation. We augment FTP with checksum checks. -->
      <retries>3</retries>
    </remoteFile>
  </common>
</settings>
```

It is possible to use more than one FTP server, but each application can only use one. The FTP server that is used for a particular transfer is determined by the application that is sending a file. If you want to use more than one FTP-server, you must use different settings for serverName (e.g. FTP-server1) and possibly also the userName (e.g. ftpUser) and userPassword (e.g. ftpPassword) when starting the applications.

Using HTTP as filetransfer method, you need to reserve a HTTP port on each machine per application. You can do this by setting the `settings.common.remoteFile.port` to e.g. `5442`

The following XML shows the the corresponding syntax in the

```
settings.xml
```

file:

```
<settings>
  <common>
    <remoteFile>
      <!-- The class to use for RemoteFile objects. -->
      <class>dk.netarkivet.common.distribute.HTTPRemoteFile</class>
      <!-- Port for embedded HTTP server -->
      <port>5442</port>
    </remoteFile>
  </common>
</settings>
```

Using the HTTPS file transfer method, you first need to generate a certificate that is used for communication. You can do this with the `keytool` application distributed with Sun Java 5 and above.

Run the following command:

```
keytool -alias NetarchiveSuite -keystore keystore -genkey
```

It should the respond with the following:

```
Enter keystore password:
```

Enter the password for the keystore.

The keytool will now prompt you for the following information

```
What is your first and last name?
  [Unknown]:
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:
```

Answer all the questions, and end with "yes".

Finally you will be asked for the certificate password.

```
Enter key password for <NetarchiveSuite>
        (RETURN if same as keystore password):
```

Answer with a password for the certificate.

You now how a file called `keystore` which contains a certificate. This keystore needs to be available for all NetarchiveSuite applications, and referenced from settings as the following example shows:

```
<settings>
  <common>
    <remoteFile>
      <!-- The class to use for RemoteFile objects. -->
      <class>dk.netarkivet.common.distribute.HTTPSRemoteFile</class>
      <!-- The port for the remote file transfers -->
      <port>8300</port>
      <!-- The keystore -->
      <certificateKeyStore>path/to/keystore</certificateKeyStore>
      <!-- The keystore passwd -->
      <certificateKeyStorePassword>testpass</certificateKeyStorePassword>
      <!-- The key password-->
      <certificatePassword>testpass2</certificatePassword>
    </remoteFile>
  </common>
</settings>
```

To keep your environment secure, you should make sure that the keystore and settings file ''only'' are readable for the user running the application.

Configure a JMS broker

The data transfer method can be configured as a plug-in (see also Appendix A - Plug-ins in NetarchiveSuite).

In the below configuration, the JMSbroker resides at `localhost`, and listens for messages on port 7676.

You must also select a JMS environment name corresponding to the `environmentName` NetarchiveSuite setting (see Common part). This allows you have more than one running installation of the NetarchiveSuite, each with its own `environmentName`. This also makes it easy to clean-up the JMS queues associated with a given `environmentName`.

The NetarchiveSuite currently only supports one kind of JMS broker, so only the 'broker','port', and 'environmentName' can be changed.

```
<settings>
  <common>
    <jms>
      <!-- Selects the broker class to be used. Must be a subclass of
            dk.netarkivet.common.distribute.JMSConnection.-->
      <class>dk.netarkivet.common.distribute.JMSConnectionSunMQ</class>
      <!-- The JMS broker host contacted by the JMS connection -->
      <broker>localhost</broker>
      <!-- The port the JMS connection should use -->
      <port>7676</port>
    </jms>
  </common>
</settings>
```

Configure Repository

The repository is configured as a simple local repository or a complex distributed repository having a distributed bitarchive replicas.

A simple repository can be configured as a plug-in using
`dk.netarkivet.common.distribute.arcrepository.LocalArcRepositoryClient` for the
`settings.common.arcrepositoryClient.class` (see also Appendix A - Plug-ins in NetarchiveSuite).

for a more complex distributed repository with at least two replicas, the settings for replicas must be defined. In this example we look at two bitarchive replicas, here called ReplicaOne and ReplicaTwo.

The following is an example of settings for a repository with two bitarchive replicas

```
<settings>
  <common>
    <replicas>
      <!-- The id's, tyes and names of all bitarchive replicas in the environment. -->
      <replica>
        <replicaId>ONE</replicaId>
        <replicaType>bitarchive</replicaType>
        <replicaName>ReplicaOne</replicaName>
      </replica>
      <replica>
        <replicaId>TWO</replicaId>
        <replicaType>bitarchive</replicaType>
        <replicaName>ReplicaTwo</replicaName>
      </replica>
    </replicas>
  </common>
</settings>
```

For applications that needs to communicate with one of the replicas, the `useReplicaId` must be set. The `useReplicaId` is used to point at which of the replicas that by default is used e.g. for execution of batch jobs – typically the Replica with the greater amount of processing power and/or minimal size of storage space per bitarchive application.

Furthermore the common replica definition should conform to settings for corresponding bitarchive applications and bitarchive monitors, i.e. the `useReplicaId` must correspond to the replica that it is representing.

```
<settings>
  <common>
    <useReplicaId>TWO</useReplicaId>
  </common>
</settings>
```

Configure job-generation

The scheduling takes place every one minute, unless the previous scheduling is not finished yet. The scheduling interval cannot be changed. Scheduling amounts to searching for active harvestdefinitions, that is ready to have jobs generated, and subsequently submitted for harvesting.

The job-generation procedure are governed by a set of settings prefixed by "settings.harvester.scheduler.". These settings rule how large your crawljobs are going to be, and how long time they will take to complete. Note that harvestdefinitions consist of at least one !DomainConfiguration, (containing a Heritrix setup, and a seed-list), and that there are two kinds: Snapshot Harvestdefinitions, and Selective Harvestdefinitions.

During scheduling, each harvest is split into a number of *crawl jobs*. This is done to keep Heritrix from using too much memory and to avoid that particularly slow or large domains cause harvests to take longer than necessary. In the job splitting part of the scheduling, the scheduler partitions a large number of DomainConfigurations into several crawljobs. Each crawljob can have only one Heritrix setup, so DomainConfigurations with different Heritrix setups will be split into different crawljobs. Additionally, a number of parameters influence what configurations are put into which jobs, attempting to create jobs that cover a reasonable amount of domains of similar sizes.

If you don't want to have the harvests split into multiple jobs, you just need to set each of

- `settings.harvester.scheduler.jobs.maxRelativeSizeDifference,`
- `settings.harvester.scheduler.jobs.minAbsoluteSizeDifference,`
- `settings.harvester.scheduler.jobs.maxTotalSize` and
- `settings.harvester.scheduler.configChunkSize`

to a large number, such as MAX_LONG. Initially, we suggest you don't change these parameters, as the way they work together is subtle. Harvests will always be split in different jobs, though, if they are based on different order.xml templates, or if different harvest limits need to be enforced.

settings.harvester.scheduler.errorFactorPrevResult: Used when calculating expected size of a harvest of some domain during the job-creation process for snapshot harvests. This defines the factor by which we maximally allow domains that have previously been harvested to increase in size, compared to the value we estimate the domain to be. In other words, it defines how conservative our estimates are. The default value is 10, meaning that the maximum number of bytes harvested is as most 10 times as great as the value we use as expected size.

settings.harvester.scheduler.errorFactorBestGuess: Used when calculating expected size of a harvest of some domain during job-creation process for a snapshot Harvests. This defines the factor by which we maximally allow domains that have previously been incompletely harvested or not harvested at all to increase in size, compared to the value we estimate the domain to be. In other words, it defines how conservative our estimates are. The default value is 20, meaning that the maximum number of bytes harvested is as most 20 times as great as the value we use as expected size. This is probably an unreasonable number, it should be reset to 2 for most installations.

settings.harvester.scheduler.expectedAverageBytesPerObject: How many bytes the average object is expected to be on domains where we don't know any better. This number should grow over time, as of end of 2005 empirical data shows 38000. Default is 38000.

settings.harvester.scheduler.maxDomainSize: Initial guess of #objects in an unknown domain. Default value is 5000

settings.harvester.scheduler.jobs.maxRelativeSizeDifference: The maximum allowed relative difference in expected number of objects retrieved in a single job definition. Set to MAX_LONG for no splitting.

settings.harvester.scheduler.jobs.minAbsoluteSizeDifference: Size differences for jobs below this threshold are ignored, regardless of the limits for the relative size difference. Set to MAX_LONG for no splitting. Default value is 2000.

settings.harvester.scheduler.jobs.maxTotalSize: When this limit is exceeded no more configurations may be added to a job. Set to MAX_LONG for no splitting. Default value is 2000000

settings.harvester.scheduler.configChunkSize: How many domain configurations we will process in one go before making jobs out of them. This amount of domains will be stored in memory at the same time. Set to MAX_LONG for no job splitting. The default value is 10000.

MAX_LONG refers to the number 2^63-1 or 9223372036854775807.

Configure Domain Granularity

The NetarchiveSuite software is bound to the concept of Domains, where a Domain is defined as

```
"domainname"."tld"
```

This concept is useful for grouping harvests with regard to specific domains.

It can be configured what is considered a TLD by changing the settings files. The settings file currently distributed with the NetarchiveSuite software will list all country-level top-level-domains as "tld"s like ".dk", ".se" and ".no". However, as a proof of concept, for ".uk"-domains, there is listed the pseudo-top-level-domains ".co.uk", ".gov.uk", ".edu.uk" and some more.

Currently, only grouping by domain suffix is supported, see NAS-1637 Plugin of Domain definition suggested.

Configure Heritrix process

In this section the configuration for running Heritrix processes via NetarchiveSuite is described. For details on managing heritrix harvest templates (order.xml), please refer to Appendix B - Managing Heritrix Harvest Templates (order.xml).

The communication between NetarchiveSuite and Heritrix is handled by the settings.harvester.harvesting.heritrixController.class plugin (see also

Appendix A - Plug-ins in NetarchiveSuite). However, only one supported implementation is bundled with NetarchiveSuite, the JMXHeritrixController.

Each harvester runs an instance of Heritrix for each harvest job being executed. It is possible to get access to the Heritrix web user interface for purposes of pausing or stopping a job, examining details of an ongoing harvest or even, if necessary, change an ongoing harvest. Note that some changes to harvests, especially those that change the scope and limits, may confuse the harvest definition system. We suggest using the Heritrix UI only for examination and pausing/terminating jobs.

Each harvest "application" running requires two ports,

- one for the user interface
. The user interface port is set by the `settings.harvester.harvesting.heritrix.guiPort` setting, and should be open to the machines that the user interface should be accessible from. Make sure to have different ports for each harvest application if you're running more than one on a machine. Otherwise, your harvest jobs will fail when two harvest applications happen to try to run at the same time – an error that could go unnoticed for a while, but which is more likely to happen exactly in critical situations where more harvesters are needed.
- one for JMX (Java Management Extensions which communicates with Heritrix).
. The JMX port is set by the `settings.harvester.harvesting.heritrix.jmxPort` setting, and does not need to be open to other machines.

The Heritrix user interface is accessible through a browser using the port specified, e.g. http://my.harvester.machine:8090, and entering the administrator name and password set in the `settings.harvester.harvesting.heritrix.adminName` and `settings.harvester.harvesting.heritrix.adminPassword` settings.

In order for the harvester application to communicate with Heritrix there need to be a username and password for the JMX controlRole which is used for this communication. This username and password must be in the settings `settings.harvester.harvesting.heritrix.jmxUsername` and `settings.harvester.harvesting.heritrix.jmxPassword`. These also need to be inserted for the corresponding values in the `conf/jmxremote.password` file (template [examples/jmxremote_template.password|https://sbforge.org/svn/netarchivesuite/trunk/examples/jmxremote_template.password]). Here you find has them on line

```
controlRole JMX_CONTROL_ROLE_PASSWORD_PLACEHOLDER.
```

Example of the above mentioned settings is given here:

```
<settings>
    <harvester>
        <harvesting>
            <heritrix>
                <adminName>admin</adminName>
                <adminPassword>adminPassword</adminPassword>
                <guiPort>8090</guiPort>
                <jmxPort>8091</jmxPort>
                <jmxUsername>controlRole</jmxUsername>
                <jmxPassword>JMX_CONTROL_ROLE_PASSWORD_PLACEHOLDER</jmxPassword>
            </heritrix>
        </harvesting>
    </harvester>
</settings>
```

It is also possible to use JConsole to access the JMX interface of the Heritrix process.

The final setting for the Heritrix processes is the amount of heap space each process is allowed to use. Since Heritrix uses a significant amount of heap space for seen URLs and other stuff, it is advisable to keep the `settings.harvester.harvesting.heritrix.heapSize` setting at at least its default setting of 1.5G if there is enough memory in the machine for this (remember to factor in the number of harvesters running on the machine – swapping will slow the crawl down "significantly").

Configure web page look

The look of the web pages can be changed by changing files in the webpages directory. The files are distributed in war-files, which are simply zip-files. They can be unpacked to customize styles, and repacked afterwards using zip. Each of the five war files under webpages corresponds to one section of the web site, as seen in the left-hand menu. The two PNG files transparent_logo.png and transparent_menu_logo.png are used on the front page and atop the left-hand menu, respectively. They can be altered to suite your whim, but the width of transparent_menu_logo.png should not be increased so much that the menu becomes overly wide. The color scheme for each section is set in the netarkivet.css file for that section and can be changed to suit your whim, though we recommend changing them all at the same time to provide a uniform look.

Configure security

Security in NetarchiveSuite is mainly defined in the `examples/security_template.policy` file. This file controls two main configurations:

Which classes are allowed to do anything (core classes), and which classes are only allowed to read the files in the bit archive (third-party batch classes). It is recommended that you fit this template to your own requirements, and store in a CVS/SVN repository locally, as we do at the Netarkivet.

To enable the use of the security policy, you will need to launch your applications with the command line options `-Djava.security.manager` and `-Djava.security.policy=examples/security_template.policy`. (Note: In NetarchiveSuite 3.8.*, the bundled security template was placed in `conf/` and named `security.policy`).

Core classes

For the core classes, we need to identify all the classes that can be involved. The default security.policy file assumes that the program is started from the root of the distribution. If that is not the case, the codeBase entries must be changed to match. The following classes should be included:

- The dk.netarkivet.* jar files and supporting jar files, located in the `lib` directory. By default, all files in this directory and its subdirectories are included by the statement

```
grant codeBase "file:lib/-" {
  permission java.security.AllPermission;
};
```

- The heritrix jar files and supporting jar files for it, usually located in the `lib/heritrix/lib` directory. By default, these are included by the above.
- The standard Java classes, which by default are included by the statement

```
grant codeBase "file:${java.home}/-" {
  permission java.security.AllPermission;
};
```

- The classes compiled by JSP as part of the web interface. These classes only exist on the machine(s) that run a web interface, and are found in the directory specified by the `settings.common.tempDir` setting. The default security file contains entries that assume this directory is `tests/commontempdir`. Note that an entry is required for each section of the web site:

```
grant codeBase "file:tests/commontempdir/Status/jsp/-" {
  permission java.security.AllPermission;
};
```

If you change the `settings.common.tempDir` setting, you will need to change this entry, too, or the web pages won't work.

Third-party classes

The default security.policy file includes settings that allow third-party batch jobs to read the bitarchives set up for the [Quick Start Manual 3.16] system. In a real installation, the bitarchive machines must specify which directories should be accessible and set up permissions for these. The default setup is:

```
grant {
  permission java.util.PropertyPermission "settings.archive.bitarchive.useReplicaId", "read";
  permission java.io.FilePermission
"${user.home}/netarchive/scripts/simple_harvest/bitarchive1/baseFileDir/*", "read";
  permission java.io.FilePermission
"${user.home}/netarchive/scripts/simple_harvest/bitarchive2/baseFileDir/*", "read";
};
```

Notice how these permissions are not granted to a specific codebase, but the permissions given are very restrictive: The classes can read files in two explicitly stated directories, and can query for the value of the `settings.archive.bitarchive.useReplicaId` setting – all other settings are off-limits, as is reading and writing other files, including temporary files. If you wish to allow third-party batch jobs to do more, think twice first – loopholes can be subtle.

Configure monitoring (allocating JMX and RMI ports)

Monitoring the deployed NetarchiveSuite relies on JMX (Java Management Extensions). Each application in the NetarchiveSuite needs its own

JMX-port and associated RMI-port, so they can be monitored from the NetarchiveSuite GUI with the StatusSiteSection, and using **jconsole** (see below). You need to select a range for the JMX-ports. In the example below, the chosen JMX/RMI-range begins at 8100/8200. It is important that no two applications on the same machine use the same JMX and RMI ports!

On each machine you need to set the JMX and RMI ports, using the settings `settings.common.jmx.port` and `settings.common.jmx.rmiPort`.

Firewall Note: This requires that the admin-machine has access to each machine taking part in the deployment on ports 8100-8300.

JMX roles

You need to select a username and password for the monitor JMX settings. This username and password must be updated in the settings `settings.monitor.jmxUsername` and `settings.monitor.jmxPassword`.

The applications which uses Heritrix (the Harvester) need to have the username and password for the Heritrix JMX settings. This username and password must be updated in the settings `settings.harvester.harvesting.heritrix.jmxUsername` and `settings.harvester.harvesting.heritrix.jmxPassword`.

These username and password values must be inserted in the `conf/jmxremote.password` file and the `conf/jmxremote.access` file. A template for these files is placed in the examples directory (jmxremote_template.access and jmxremote_template.password)

Currently, all applications **must** use the same password.

The applications will automatically register themselves for monitoring at the GUI application, if the StatusSiteSection is deployed. All important log messages (Log level INFO and above) can be studied in the GUI. However, only the last 100 messages from each application instance are available. This number can be increased or decreased using the setting `settings.monitor.logging.historySize`.

Example:

```
<settings>
  <common>
    <jmx>
      <port>8100</port>
      <rmiPort>8200</rmiPort>
    </jmx>
  </common>
  <monitor>
    <jmxUsername>monitorRole</jmxUsername>
    <jmxPassword>JMX_MONITOR_ROLE_PASSWORD_PLACEHOLDER</jmxPassword>
    <logging>
      <historySize>100</historySize>
    </logging>
  </monitor>
  <harvester>
    <harvesting>
      <heritrix>
        <jmxUsername>controlRole</jmxUsername>
        <jmxPassword>JMX_HERITRIX_ROLE_PASSWORD_PLACEHOLDER</jmxPassword>
      </heritrix>
    </harvesting>
  </harvester>
</settings>
```

These will give the following settings in the jmxremote.password file:

```
monitorRole JMX_MONITOR_ROLE_PASSWORD_PLACEHOLDER
controlRole JMX_HERITRIX_ROLE_PASSWORD_PLACEHOLDER
```

And the following privileges in the jmxremote.access file:

```
monitorRole readonly
controlRole readwrite
```

Configure ArcRepository and BitPreservation Database

The ArcRepositoryApplication and the BitPreservation actions available from the GUIApplication can either use files or a database. As default it is set to use files, but it can be changed to database with the following settings:

```
<settings>
    <archive>
        <admin>
            <class>dk.netarkivet.archive.arcrepositoryadmin.DatabaseAdmin</class>
            <database>
                <class>dk.netarkivet.archive.arcrepositoryadmin.DerbyServerSpecifics</class>
                <baseUrl>jdbc:derby</baseUrl>
                <machine>localhost</machine>
                <port>1527</port>
                <dir>adminDB</dir>
            </database>
        </admin>
        <bitpreservation>
            <baseDir>bitpreservation</baseDir>

<class>dk.netarkivet.archive.arcrepository.bitpreservation.DatabaseBasedActiveBitPreservation</class>
        </bitpreservation>
    </archive>
</settings>
```

These parameters will give the following database URL: jdbc:derby://localhost:1527/archiveDB

If a specific URL is wanted (e.g. another database type than derby), then it should be assigned to the baseUrl and the 'machine', the 'port' and the 'dir' should be set to the empty string, e.g.:

```
<settings>
    <archive>
        <admin>
            <database>
                <class>dk.netarkivet.archive.arcrepositoryadmin.DerbyServerSpecifics</class>
                <baseUrl>jdbc:derby://localhost:1527/adminDB</baseUrl>
<machine></machine>
                <port></port>
                <dir></dir>
            </database>
        </admin>
</settings>
```

It requires a database installed in the directory 'archiveDB' under the installation directory on the machine containing both the ArcRepositoryApplication and the GUIApplication.

If the database is to be installed through Deploy the following parameter should be added to the relevant deployMachine entity:

```
<globalBitpreservationDatabaseDir>archiveDB</globalBitpreservationDatabaseDir>
```

Examples of deploy configuration files

The following example of configuration file requires adaptation to your own system before use.

deploy_distributed_example.xml

The instance with two replicas divided over two physical locations Each physical locations contain several machines Bitarchive machines, harvester machine and viewerproxy machine Only one physical location has an administator machine, which contains the GUI application, the Bitarchive monitors, the HarvestJobManager, HarvestJobMonitor and the arc repository.

Basics

Deploy Configurations

Deploy Configurations

Contents

The Deploy module can make the scripts for deployment, installation, start and stop of the !NetarchiveSuite applications. In order to use this module, it is necessary to make an special configuration file containing settings for the applications as well as special deploy settings. For more information please refer to the Installation Manual.

Detailed Configurations

Heritrix Configurations

Heritrix Configurations

Contents

- How to configure which Heritrix report has to be uploaded in the metadata ARC file

For configuration related to NetarchiveSuite, please refer to section on Detailed Configurations#Configure Heritrix process.

For more specific Heritrix configurations, please refer to Appendix B - Managing Heritrix Harvest Templates (order.xml) and Appendix C - Migrate the Heritrix templates to NetarchiveSuite 3.6.0+ of this document.

The crawling in NetarchiveSuite uses by default Deduplication. This feature and how to disable it is described in Configuration Manual, Section 8.1.2.

How to configure which Heritrix report has to be uploaded in the metadata ARC file

Three settings properties control which heritrix reports are added to the metadata ARC file:

- **settingsharvesterharvestingmetadataheritrixFilePattern** is a java pattern that allows you select which files in the crawl dir (not recursively) to include in the metadata ARC.
- **settingsharvesterharvestingmetadatareportFilePattern** is also a java pattern that controls which subset of the files selected by heritrixFilePattern are to be considered as report files All the other files will be considered as setup files.
- **settingsharvesterharvestingmetadatalogFilePattern** is a third java pattern that controls which files in the logs subdirectory of the crawldir are to be added as log files to the metadata ARC.

Deploy Configurations

Wayback Configuration

Wayback Configuration

Contents

- Requirements
- Configuration
    - wayback.xml

The Wayback installation under NetarchiveSuite is only tested on a pc installed with linux and in ProxyReplay mode. Other modes should work, but no guaranties are given.

Requirements

The following applications should be running and reachable from the machine running Tomcat with Wayback web application.

1. JMS server.
2. FTP server.
3. Archive (eg. Standalone archive given in ./conf/wayback/standalone_archive.xml). The needed applications from NetarchiveSuite is BitarchiveApplication, BitarchiveMonitorApplication, ArcRepositoryApplication. The NetarchveSuite version should be newer than 3.10. This setup has been tested with Tomcat (6.0.20).

When configuring Wayback to work with NetarchiveSuite, the above services is needed, furthermore it is needed to have a full source package of the NetarchiveSuite and an installation of **ant**, it has been tested with 1.7.1.

Configuration

The two configuration files that should be modified are located in **./conf/wayback/** in the NetarchiveSuite full source package. The files are named **CDXCollection.xml** and **wayback.xml**.

wayback.xml

In this config file there are multiple settings that should be changed to fit your setup, to make the system run correctly:

wayback.basedir=/tmp/wayback - The web application should have read and write access to this directory.

The port should be specified in the following three lines, and be available (i.e. not yet already used by another application).

- <bean name="8080:wayback" class="org.archive.wayback.webapp.AccessPoint">
- <property name="replayURIPrefix" value="http://localhost.archive.org:8080/wayback/"/>
- <bean name="8090" parent="8080:wayback">

CDXCollection.xml

This configuration file describes where Wayback finds its CDX files (i.e indices of the ARC/WARC files).

In this file it should only be necessary to change the following path to point a local CDX collection.

<value>/wayback/file.sorted.cdx</value>

Compiling Tomcat target

This can be done from the NetarchiveSuite root directory. By running the command **ant -file wayback.build.xml warfile**, this produces a ROOT.war file in the NetarchiveSuite root director, and this ROOT.war file should be copied to* $TOMCAT_HOME/webapps/*.

Tomcat should furthermore have access to a settings.xml file, see below. This can be done by adding the following line to **$TOMCAT_HOME/bin/catalina.sh** just after the first line.

*CATALINA_OPTS='-Ddk.netarkivet.settings.file=$TOMCAT_HOME/webapps/ROOT/WEB-INF/settings.xml' *

This setting file is a NetarchiveSuite settings.xml file, and only includes the **common** and **wayback** sections.

The following settings should be modified to fit the local installation.

Change the following to match the FTP settings on the system.

```
        <remoteFile>
            <-- TODO: See user documentation for NetarchiveSuite
            http://netarkivet.dk/suite/Documentation . -->
    <serverName>ftp.yourdomain.com</serverName>
            <userName>ftpuser</userName>
            <userPassword>ftppassword</userPassword>
        </remoteFile>
```

Update the following mail settings

```
<mail>
    <server>mail.yourdomain.com</server>
</mail>
<notifications>
    <class>dk.netarkivet.common.utils.EMailNotifications</class>
    <sender>example@yourdomain.com</sender>
    <receiver>example@yourdomain.com</receiver>
</notifications>
```

Described elsewhere

It is outside the scope of this configuration guide to describe how to harvest a ARC/WARC file. It is also outside the scope of this guide to describe how to get import an ARC/WARC collection into Wayback by way of CDX-entries for each object in the collection.

Setting up NetarchiveSuite archive is described elsewhere and a sample setup file is given in the NetarchiveSuite source package.

Heritrix Configurations

Configuring External Software

Configuring External Software

Contents

- Configuring a JMS broker
- Configuring FTP

Configuring a JMS broker

Please refer to JMS section in Installation Manual.

Configuring FTP

Please refer to FTP section in Installation Manual.

Wayback Configuration

BatchGUI

BatchGUI

Contents

The BatchGUI is the user interface for executing NetarchiveSuite-wrapped batchjobs for performing datamining on the archive.
It is currently located under the Bitpreservation-sitesection of the web-interface for NetarchiveSuite.

To be able to access these batchjobs, the system must be aware of which batchjobs are available. This is done through the settings of the GUIApplication.
A batchjob for the BatchGUI is defined by the class of the batchjob and the jar-file where it is located, and it is defined in the settings file under settings.common, where the default is the following:

```
<settings>
    <common>
        <batch>
            <batchjobs>
                <batchjob>
                    <class>dk.netarkivet.common.utils.batch.ChecksumJob</class>
                    <jarfile/>
                </batchjob>
                <batchjob>
                    <class>dk.netarkivet.common.utils.batch.FileListJob</class>
                    <jarfile/>
                </batchjob>
            </batchjobs>
        </batch>
    </common>
</settings>
```

Note that the default batchjobs does not have any specified jarfile, since they are part of the common-module in NetarchiveSuite and thus available for every application.

For adding another batchjob, you just need to define the class of the batchjob and the path to the jar-file (from the installation directory).
E.g. if you have a batchjob for retrieving the mimetypes, which has the classpath batchprogs.Mimetypes are located in a jar-file called batch.jar located in the directory externals, then you add the following to the settings:

```
<batchjob>
            <class>batchprogs.Mimetypes</class>
            <jarfile>externals/batch-mime.jar</jarfile>
        </batchjob>
```

The example can be found here:
batch-mime.jar

If any errors or typos are within this settings, the BatchGUI will inform you about the problem when you look at the page.

Configuring External Software

Appendix_A

Appendix A - Plug-ins in NetarchiveSuite

Contents

All the settings above ending on ".class" indicate that the implementation of a certain feature can be replaced by alternative implementations. There is usually a choice of several classes to choose from. But our framework does at least enable the installer to replace the default class with a class of his own, if no existing alternatives are suitable.

We now describe the available plugs, and existing plugins for these plugs.

settings.common.remoteFile.class: This setting allows you to select your chosen way of filetransfer in the NetarchiveSuite. You can here choose between FTPRemoteFile (where the data is transferred using a FTP-server), HTTPRemoteFile (where the data is transferred using a two embedded webservers (one at each end), and HTTPSRemoteFile which works just like HTTPRemoteFile except it uses a shared certificate file for secure communication. Note that the HTTPRemoteFile and HTTPSRemoteFile requires dedicated ports in the firewall to be open between all possible senders and recipients of data. For implementers of new filetransfer methods, this class must implement the class dk.netarkivet.common.distribute.RemoteFile. The default value is FTPRemoteFile.

settings.harvester.datamodel.database.specifics.class: This setting allows you select which type of database you want to use. There are support for 3 types already: An Embedded Derby database (dk.netarkivet.harvester.datamodel.DerbyEmbeddedSpecifics), an external Derby database (dk.netarkivet.harvester.datamodel.DerbyClientSpecifics), or an MySQL database (dk.netarkivet.harvester.datamodel.MySQLSpecifics). The default is DerbyEmbeddedSpecifics. If you choose not to use the default, you need to replace the default database URL(setting settings.harvester.datamodel.database.url), and maybe the time for the daily backup to start (setting settings.harvester.datamodel.database.backupInitHour))

settings.common.jms.class This class designates what kind of JMS broker the NetarchiveSuite uses to send messages between applications. Presently only the Sun JMS brokers is supported (dk.netarkivet.common.distribute.JMSConnectionSunMQ). This class must implement the dk.netarkivet.common.distribute.JMSConnection class.

settings.common.arcrepositoryClient.class. Must implement dk.netarkivet.common.distribute.ArcRepositoryClient The available choices are the default dk.netarkivet.archive.arcrepository.distribute.JMSArcRepositoryClient (that is required, if you want to access the distributed type of archive that is included in the NetarchiveSuite). and the dk.netarkivet.common.distribute.LocalArcRepositoryClient (allows for access to a local archive)

settings.common.notifications.class: Allows for different ways of making notifications. The default choice is the class

```
dk.netarkivet.common.utils.EMailNotifications
```

(which allows you to receive notifications by email). The use of this plugin requires setting the mail-server, the recipient- and sending email-address. Alternatively, you can use

```
dk.netarkivet.common.utils.PrintNotifications
```

, which simply prints the notifications to

```
stderr
```

on the terminal.

settings.common.webinterface.sitesection.class This setting allows you to add webmodules to the NetarchiveSuite GUI. Several SiteSection classes can be active in the same GUI. the default(standard) configuration contains all 5 existing webmodules:

1. HarvestDefinition: Allows you to define and schedule harvests ,
2. HarvestHistory: See the status of running and finished harvestjobs
3. BitPreservation: This module has tools for sanity testing data in the bitarchives
4. QA: Module for doing Quality Assurance
5. Status: Module for monitoring the health of all machines and applications

settings.common.webinterface.language: The languages supported by the webinterface. Danish (locale=da), English (locale=en), German (locale=de), and Italian (locale=it) are supported currently. The Coding Guidelines will tell you how to add support for more languages to the NetarchiveSuite.

settings.common.indexClient: The client selected for access to indices. Indices are requested by the HarversterControllerApplication instances.

```
    <indexClient>
        <!-- The class instantiated to give access to indices. Will be created by IndexClientFactory
  -->
        <class>dk.netarkivet.archive.indexserver.distribute.IndexRequestClient</class>
        <!-- The amount of time, in milliseconds, we should wait for replies
            when issuing a call to generate an index over som jobs.
         -->
        <indexRequestTimeout>43200000</indexRequestTimeout>
    </indexClient>
```

settings.common.monitorregistryClient.class. This defines which class to use for monitor registry. Must implement the interfacedk.netarkivet.common.distribute.monitorregistry.MonitorRegistryClient. There is two available implementations:

- dk.netarkivet.common.distribute.monitorregistry.PrintMonitorRegistryClient (just prints out how to stdout the JMXport and RMIport to use for connecting to its JVM.
- dk.netarkivet.monitor.distribute.JMSMonitorRegistryClient: registers itself centrally with a registry by sending JMS messages every minute. This delay can be configured with the *settings.common.monitorregistryClient.reregisterdelay *setting.
  The default class is dk.netarkivet.monitor.distribute.JMSMonitorRegistryClient. **settings.common.freespaceprovider.class**: This setting defines which plugin to use for reporting how much free space is available. Must implement the `dk.netarkivet.common.utils.FreeSpaceProvider` interface. Available implementations are:

- dk.netarkivet.common.utils.DefaultFreeSpaceProvider (uses File.getUsableSpace() to compute the free space available)
- dk.netarkivet.common.utils.FilebasedFreeSpaceProvider (Reads the free space available out of a file)
  The default class is dk.netarkivet.common.utils.DefaultFreeSpaceProvider.

settings.archive.admin.class: Class for accessing and manipulating the administrative data for the ArcRepository. All classes must implement the dk.netarkivet.archive.arcrepositoryadmin.AdminData interface. The available implementations are:

- dk.netarkivet.archive.arcrepositoryadmin.UpdateableAdminData (filebased implementation that uses a admin.data file containing the ingested files and their checksums).
- dk.netarkivet.archive.arcrepositoryadmin.DatabaseAdmin (database implementation that uses a database defined by the following settings: settings.archive.admin.database.[class|machine|port|dir].
  The default class is dk.netarkivet.archive.arcrepositoryadmin.UpdateableAdminData

settings.archive.admin.database.class: Which class to use for your adminDB database. This plugin is used, if the setting settings.archive.admin.class is set to the class `dk.netarkivet.archive.arcrepositoryadmin.DatabaseAdmin` and the setting settings.archive.bitpreservation.class is set to the class `dk.netarkivet.archive.arcrepository.bitpreservation.DatabaseBasedActiveBitPreservation`.

settings.archive.bitpreservation.class: Setting for which class should handle ActiveBitPreservation. All implementations must implement the `dk.netarkivet.archive.arcrepository.bitpreservation.ActiveBitPreservation`. The following implementations are available:

- dk.netarkivet.archive.arcrepository.bitpreservation.DatabaseBasedActiveBitPreservation (uses a database to store the results of the bitpreservation actions).
- dk.netarkivet.archive.arcrepository.bitpreservation.FileBasedActiveBitPreservation (stores the results of bitpreservation actions to a set of files on disk.)

settings.harvester.harvesting.heritrixController.class: This class handles the communication to a running Heritrix instance. All implementations must implement the `dk.netarkivet.harvester.harvesting.HeritrixController` interface. There are two implementations available of which one is deprecated:

- dk.netarkivet.harvester.harvesting.DirectHeritrixController (deprecated; embeds a Heritrix CrawlController which starts and stops one crawl job)
- dk.netarkivet.harvester.harvesting.JMXHeritrixController (Starts Heritrix as an independent process, ready to crawl a predefined crawljob; Heritrix is asked to shutdown after the crawljob has terminated)
  The default class is `dk.netarkivet.harvester.harvesting.JMXHeritrixController`.

settings.wayback.urlcanonicalizer.classname: The class used to canonicalize urls. This class must implement the interface org.archive.wayback.UrlCanonicalizer. The only acceptable implementation is `dk.netarkivet.wayback.batch.copycode.NetarchiveSuiteAggressiveUrlCanonicalizer` which is luckily the default class.

BatchGUI

Appendix_B

Appendix B - Managing Heritrix Harvest Templates (order.xml)

Contents

The NetarchiveSuite software uses Heritrix 1.14.4 to harvest webpages. A harvest done by Heritrix is specified with a harvest template (invariably named order.xml). A harvest template describes how much to harvest and from where. Furthermore a seedlist is always associated with a given order.xml.

The standard harvest template used by NetarchiveSuite follow the order.xml standard of Heritrix 1.10+.

Our default harvest template can be seen here in full: default_orderxml.xml

If you intend to build your own templates, it is recommended to use this template as a baseline.

Mandatory elements in the NetarchiveSuite and their role

A number of elements in the order.xml are required in all NetarchiveSuite harvest templates:

A. The QuotaEnforcer

The QuotaEnforcer is used to restrict the number of bytes harvested from each domain in the seedlist.

```xml
<newObject name="QuotaEnforcer" class="org.archive.crawler.prefetch.QuotaEnforcer">
        <boolean name="force-retire">false</boolean>
        <boolean name="enabled">true</boolean>
        <newObject name="QuotaEnforcer#decide-rules" class=
"org.archive.crawler.deciderules.DecideRuleSequence">
                <map name="rules">
                </map>
        </newObject>
        <long name="server-max-fetch-successes">-1</long>
        <long name="server-max-success-kb">-1</long>
        <long name="server-max-fetch-responses">-1</long>
        <long name="server-max-all-kb">-1</long>
        <long name="host-max-fetch-successes">-1</long>
        <long name="host-max-success-kb">-1</long>
        <long name="host-max-fetch-responses">-1</long>
        <long name="host-max-all-kb">-1</long>
        <long name="group-max-fetch-successes">-1</long>
        <long name="group-max-success-kb">-1</long>
        <long name="group-max-fetch-responses">-1</long>
        <long name="group-max-all-kb">-1</long>
        <boolean name="use-sparse-range-filter">true</boolean>
</newObject>
```

B. The DeDuplicator

The DeDuplicator is a module authored by Kristinn Sigurdsson from the National Library of Iceland. It is part of the Write-processor chain. It enables us to avoid saving duplicates in our storage. It does this by looking up the url of the potential duplicate object in the index associated with this module. If the url is found in the index, and the checksum for the url in the index is unaltered, the object is not stored. However a reference to where the object is stored is written to the crawl log. If the url for the object is not found in the index, the object is stored normally. Note that only non-text objects are examined by this module, i.e. where the mimetype of the object does not match "^text/." **(like text/html or text/plain). Note that the deduplication is disabled if either the DeDuplicator element in the harvest template is disabled (the value of the attribute "enabled" is set to false), or the general setting *settings.harvester.harvesting.deduplication.enabled** is set to **false**. NetarchiveSuite uses version 0.4.0 of the deduplicator.

```xml
<newObject name="DeDuplicator" class="is.hi.bok.deduplicator.DeDuplicator">
        <boolean name="enabled">true</boolean>
        <map name="filters">
        </map>
        <string name="index-location"/>
        <string name="matching-method">By URL</string>
        <boolean name="try-equivalent">true</boolean>
        <boolean name="change-content-size">false</boolean>
        <string name="mime-filter">^text/.*</string>
        <string name="filter-mode">Blacklist</string>
        <string name="analysis-mode">Timestamp</string>
        <string name="log-level">SEVERE</string>
        <string name="origin"/>
        <string name="origin-handling">Use index information</string>
        <boolean name="stats-per-host">true</boolean>
</newObject>
```

C. The "http-headers" element

This element describes, how Heritrix will present itself to the webservers when fetching data. It points by default to the non-existing webpage

http://my_website.com/my_infopage.html and the equally non-existing mail address " my_email@my_website.com ". Please update this to your own institution and email

```
        <map name="http-headers">
            <string name="user-agent">Mozilla/5.0 (compatible; heritrix/1.14.3 +http:
//my_website.com/my_infopage.html)</string>
<string name="from">my_email@my_website.com</string>
        </map>
```

D. The Archiver element

This element does the actual writing of the fetched objects to an arcfile. In the future we may want to write to WARC files instead, which can be easily be done. Heritrix allows you to have multiple 'Writers' in use at the same time. For instance, you can write your objects to both ARC and WARC at the same time, as well as writing the objects to a database.

```
<newObject name="Archiver" class="org.archive.crawler.writer.ARCWriterProcessor">
                <boolean name="enabled">true</boolean>
                <newObject name="Archiver#decide-rules" class=
"org.archive.crawler.deciderules.DecideRuleSequence">
                    <map name="rules">
                    </map>
                </newObject>
                <boolean name="compress">false</boolean>
                <string name="prefix">IAH</string>
                <string name="suffix">${HOSTNAME}</string>
                <integer name="max-size-bytes">100000000</integer>
                <stringList name="path">
                    <string>arcs</string>
                </stringList>
                <integer name="pool-max-active">5</integer>
                <integer name="pool-max-wait">300000</integer>
                <long name="total-bytes-to-write">0</long>
                <boolean name="skip-identical-digests">false</boolean>
        </newObject>
```

E. The ContentSize element

To have statistics work right when jobs finishes and goes back into the database all templates in NetarchiveSuite require a special content-size annotation post-processor. If this element is not present, the size will allways be 0 in the database for harvests done without this in the template:

```
<newObject name="ContentSize"
class="dk.netarkivet.harvester.harvesting.ContentSizeAnnotationPostProcessor">
                <boolean name="enabled">true</boolean>
                <newObject name="ContentSize#decide-rules"
class="org.archive.crawler.deciderules.DecideRuleSequence">
                    <map name="rules">
                    </map>
                </newObject>
        </newObject>
```

F. The Scope element

The scope element decides which urls to harvest and which not to harvest. Before release 3.6.0, we used the following three scopes:

A. DomainScope. The standard NetarchiveSuite scope allows the harvester to fetch all objects coming from any 2nd level domains represented by one of the seeds. Embeddded objects, like images, and stylesheets are always fetched even when coming from other domains.
A. HostScope. This scope are restricted to fetching objects from the hosts represented by the seeds.
A. PathScope. This scope are restricted to fetching objects from

These 3 scopes were all deprecated from Heritrix 1.10.0, and now all NetarchiveSuite templates are required to use the DecidingScope instead. This type of Scope uses a sequence of DecideRules to define the scope of the harvest. We now emulate these three scopes by adding a specific DecideRule to the DecidingScope. In the case of DomainScope, it required designing our own DecideRule (

`dk.netarkivet.harvester.harvesting.OnNSDomainsDecideRule`). So for DomainScope type scopes, you add the following element:

```xml
<newObject name="acceptURIFromSeedDomains" class=
"dk.netarkivet.harvester.harvesting.OnNSDomainsDecideRule">
                         <string name="decision">ACCEPT</string>
                         <string name="surts-source-file">seeds.txt</string>
                         <boolean name="seeds-as-surt-prefixes">false</boolean>
                         <string name="surts-dump-file"/>
                         <boolean name="also-check-via">false</boolean>
                         <boolean name="rebuild-on-reconfig">true</boolean>
</newObject>
```

Emulating the HostScope requires adding the OnHostsDecideRule element:

```xml
<newObject name="acceptIfOnSeedsHosts" class="org.archive.crawler.deciderules.OnHostsDecideRule">
                         <string name="decision">ACCEPT</string>
                         <string name="surts-dump-file"></string>
                         <boolean name="also-check-via">false</boolean>
                         <boolean name="rebuild-on-reconfig">true</boolean>
                </newObject>
```

Emulating the PathScope requires adding the SurtPrefixesDecideRule element:

```xml
<newObject name="acceptIfSurtPrefixed" class="org.archive.crawler.deciderules.SurtPrefixedDecideRule">
                         <string name="decision">ACCEPT</string>
                         <string name="surts-source-file"></string>
                         <boolean name="seeds-as-surt-prefixes">true</boolean>
                         <string name="surts-dump-file"></string>
                         <boolean name="also-check-via">false</boolean>
                         <boolean name="rebuild-on-reconfig">true</boolean>
                </newObject>
```

An example of a complete DecidingScope element is shown below.

```xml
        <newObject name="scope" class="org.archive.crawler.deciderules.DecidingScope">
            <boolean name="enabled">true</boolean>
            <string name="seedsfile">seeds.txt</string>
            <boolean name="reread-seeds-on-config">true</boolean>
            <!-- DecideRuleSequence. Multiple DecideRules applied in order with last non-PASS the
resulting decision -->
            <newObject name="decide-rules" class="org.archive.crawler.deciderules.DecideRuleSequence">
                <map name="rules">
                    <newObject name="rejectByDefault" class=
"org.archive.crawler.deciderules.RejectDecideRule"/>
                    <newObject name="acceptURIFromSeedDomains" class=
"dk.netarkivet.harvester.harvesting.OnNSDomainsDecideRule">
                        <string name="decision">ACCEPT</string>
                        <string name="surts-source-file"></string>
                        <boolean name="seeds-as-surt-prefixes">true</boolean>
                        <string name="surts-dump-file"/>
                        <boolean name="also-check-via">false</boolean>
                        <boolean name="rebuild-on-reconfig">true</boolean>
                    </newObject>
                    <newObject name="rejectIfTooManyHops" class=
"org.archive.crawler.deciderules.TooManyHopsDecideRule">
                        <integer name="max-hops">25</integer>
                    </newObject>
                    <newObject name="rejectIfPathological" class=
"org.archive.crawler.deciderules.PathologicalPathDecideRule">
                        <integer name="max-repetitions">3</integer>
                    </newObject>
                    <newObject name="acceptIfTranscluded" class=
"org.archive.crawler.deciderules.TransclusionDecideRule">
                        <integer name="max-trans-hops">25</integer>
                        <integer name="max-speculative-hops">1</integer>
                    </newObject>
                    <newObject name="pathdepthfilter" class=
"org.archive.crawler.deciderules.TooManyPathSegmentsDecideRule">
                        <integer name="max-path-depth">20</integer>
                    </newObject>
                    <newObject name="global_crawlertraps" class=
"org.archive.crawler.deciderules.MatchesListRegExpDecideRule">
                        <string name="decision">REJECT</string>
                        <string name="list-logic">OR</string>
                        <stringList name="regexp-list">
                        <string>.*core\.UserAdmin.*core\.UserLogin.*</string>
                        <string>.*core\.UserAdmin.*register\.UserSelfRegistration.*</string>
                        <string>.*\/w\/index\.php\?title=Speci[ae]l:Recentchanges.*</string>
                        <string>.*act=calendar&amp;cal_id=.*</string>
                        <string>.*advCalendar_pi.*</string>
                        <string>.*cal\.asp\?date=.*</string>
                        <string>.*cal\.asp\?view=monthly&amp;date=.*</string>
                        <string>.*cal\.asp\?view=weekly&amp;date=.*</string>
                        <string>.*cal\.asp\?view=yearly&amp;date=.*</string>
                        .....
                        <string>.*index\.php\?iDate=.*</string>
                        <string>.*index\.php\?module=PostCalendar&amp;func=view.*</string>
                        <string>.*index\.php\?option=com_events&amp;task=view.*</string>

<string>.*index\.php\?option=com_events&amp;task=view_day&amp;year=.*</string>

<string>.*index\.php\?option=com_events&amp;task=view_detail&amp;year=.*</string>

<string>.*index\.php\?option=com_events&amp;task=view_month&amp;year=.*</string>

<string>.*index\.php\?option=com_events&amp;task=view_week&amp;year=.*</string>
                        </stringList>
                    </newObject>
                </map> <!-- end rules -->
            </newObject> <!-- end decide-rules -->
        </newObject> <!-- End DecidingScope -->
```

The anatomy of a decidingscope

Finally, we describe the rest of the components of a decidingscope element.

The header

```xml
<newObject name="scope" class="org.archive.crawler.deciderules.DecidingScope">
            <boolean name="enabled">true</boolean>
            <string name="seedsfile">seeds.txt</string>
            <boolean name="reread-seeds-on-config">true</boolean>
            <!-- DecideRuleSequence. Multiple DecideRules applied in order with last non-PASS the
resulting decision -->
            <newObject name="decide-rules" class="org.archive.crawler.deciderules.DecideRuleSequence">
                <map name="rules">
                        <newObject name="rejectByDefault" class=
"org.archive.crawler.deciderules.RejectDecideRule"/>
```

The defining deciderule

Here we have the deciderule, that defines this as either a DomainScope, a HostScope, or a PathScope

Standard harvest rules

These rules add more restrictions to the scope:

- Restrict the amount of hops allowed from any seed. Normally set to 25.
- Restrict the amount of repetitions in a URL-path, eg. repetition/repetition/... Repetitions are normally symptoms of crawlertraps.
- Define the maximal transclusion hops, and maximal speculative hops. (
  http://crawler.archive.org/apidocs/org/archive/crawler/deciderules/TransclusionDecideRule.html)
- Restrict the maximal path depth. Normally set to 20

```xml
                        <newObject name="rejectIfTooManyHops" class=
"org.archive.crawler.deciderules.TooManyHopsDecideRule">
                                <integer name="max-hops">25</integer>
                        </newObject>
                        <newObject name="rejectIfPathological" class=
"org.archive.crawler.deciderules.PathologicalPathDecideRule">
                                <integer name="max-repetitions">3</integer>
                        </newObject>
                        <newObject name="acceptIfTranscluded" class=
"org.archive.crawler.deciderules.TransclusionDecideRule">
                                <integer name="max-trans-hops">25</integer>
                                <integer name="max-speculative-hops">1</integer>
                        </newObject>
                        <newObject name="pathdepthfilter" class=
"org.archive.crawler.deciderules.TooManyPathSegmentsDecideRule">
                                <integer name="max-path-depth">20</integer>
                        </newObject>
```

Define general crawlertraps to be avoided

Lists of crawlertraps to be avoided are defined with a MatchesListRegExpDecideRule. Here we list all crawlertraps (defined by a regular expression). If any object matches one of these regular expression, the object is not fetched (unless a previous rule require the object to be fetched).

```
<newObject name="global_crawlertraps" class=
"org.archive.crawler.deciderules.MatchesListRegExpDecideRule">
                          <string name="decision">REJECT</string>
                          <string name="list-logic">OR</string>
                          <stringList name="regexp-list">
                            <string>.*core\.UserAdmin.*core\.UserLogin.*
                          </stringList>
```

When creating a new Harvestjob, another MatchesListRegExpDecideRule is added to the harvestTemplate, that specifies the crawlertraps to be avoided.

The HarvestTemplateApplication tool

You can upload and download the templates using our GUI. This is described in our Harvester Templates. But you can also upload and download the templates using the commandline HarvestTemplateApplication. This application allows you to create, download, update templates. We have made a script to make it easier to use this application: HarvestTemplateApplication.sh.txt

```
java dk.netarkivet.harvester.tools.HarvestTemplateApplication <command> <args>
create <template-name> <xml-file for this template>
download [<template-name>]
update <template-name> <xml-file to replace this template>
showall
```

Predefined harvest templates

All our templates fall in three categories depending on the scope defined in the template. Note that our templates generally do not obey robots.txt. This is because the Danish legislation allows is to ignore the constraints dictated by robots.txt. However, there are two exceptions to this rule:

- default_obeyrobots.xml
- default_obeyrobots_withforms.xml

Even though DomainScope, HostScope, PathScope are now emulated using DecidingScope, these categories are still useful:

Templates w/ DomainScope

1. default_orderxml.xml (standard template)
2. default_withforms.xml (standard template that can handle forms)
3. default_obeyrobots.xml (standard template that can handle forms)
4. default_obeyrobots_withforms.xml (standard template that obeys robots.txt and handles forms)
5. default_orderxml_low_bandwidth.xml (standard template for sites with low bandwidth)
6. frontpages.xml (harvest template that only harvest the seeds and associated stylesheets and images)
7. frontpages_plus_1level.xml (The above plus one extra level extra)
8. frontpages_plus_2levels.xml (The above plus 2 extra levels)

Templates w/ HostScope

1. host_10levels_orderxml.xml (harvest the hosts of the seeds up to 10 levels from seeds)
2. host_100levels_orderxml.xml (harvest the hosts of the seeds up to 100 levels from seeds)

Templates w/ PathScope

1. path_10levels_orderxml.xml (harvest the hosts of the seeds up to 10 levels from seeds)
2. path_100levels_orderxml.xml (harvest the hosts of the seeds up to 100 levels from seeds)

Appendix_A

Appendix_C

Appendix C - Migrate the Heritrix templates to NetarchiveSuite 3.6.0+

Contents

If you are just using the predefined templates with few changes like changed
the email-address and website information, the easiest way to migrate is to modify the
predefined templates found in the binary distribution of NetarchiveSuite in
the harvestdefinitionbasedir/order_templates_dist directory
and change the email-adress and website information again.

If you do this, you also get the more inconsequential updates to the template:

- The removal of obsolete attributes from some elements
- Addition of new attributes to some elements

Then you just update the existing templates in your database with these modified ones using the HarvestTemplateApplication tool mentioned in
Appendix B - Managing Heritrix Harvest Templates (order.xml).
Note that some templates are no longer distributed with NetarchiveSuite. If you want to keep using
those, you need to follow the procedure described below.

If you have already put a lot effort in making your own templates, you can update your
existing templates by "only" upgrading the scope element in the templates from either a
DomainScope, HostScope, or a PathScope.

Before we explain how to migrate these scopes to a DecidingScope, you need to know something
about the anatomy of these scopes.

1) Header (includes scope class,and attributes):

```xml
<newObject name="scope" class="org.archive.crawler.scope.PathScope">
          <boolean name="enabled">true</boolean>
          <string name="seedsfile">seeds.txt</string>
          <boolean name="reread-seeds-on-config">true</boolean>
          <integer name="max-link-hops">10</integer>
          <integer name="max-trans-hops">5</integer>
```

2) An OrFilter element named "exclude-filter" containing a number of filters as components:
a HopsFilter, a PathDepthFilter, a PathologicalPathFilter, a URIRegExpFilter, a URIListRegExpFilter (filter to avoid common crawlertraps),
and potentially other types of filters:
Each of these filters will have to be converted to a similar DecideRule. Explanation to follow.

```xml
            <newObject name="exclude-filter" class="org.archive.crawler.filter.OrFilter">
                <boolean name="enabled">true</boolean>
                <boolean name="if-matches-return">true</boolean>
                <map name="filters">
                    <newObject name="hops_filter" class="org.archive.crawler.filter.HopsFilter">
                        <boolean name="enabled">true</boolean>
                    </newObject>
                    <newObject name="pathdepth" class="org.archive.crawler.filter.PathDepthFilter">
                        <boolean name="enabled">true</boolean>
                        <integer name="max-path-depth">20</integer>
                        <boolean name="path-less-or-equal-return">false</boolean>
                    </newObject>
                    <newObject name="pathologicalpath" class=
"org.archive.crawler.filter.PathologicalPathFilter">
                        <boolean name="enabled">true</boolean>
                        <integer name="repetitions">3</integer>
                    </newObject>
                    <newObject name="dr_dk" class="org.archive.crawler.filter.URIRegExpFilter">
                        <boolean name="enabled">true</boolean>
                        <boolean name="if-match-return">true</boolean>
                        <string name="regexp">.*dr\.dk.*epg\.asp.*</string>
                    </newObject>
                    <newObject name="globale_crawlertraps" class=
"org.archive.crawler.filter.URIListRegExpFilter">
                        <boolean name="enabled">true</boolean>
                        <boolean name="if-match-return">true</boolean>
                        <string name="list-logic">OR</string>
                        <stringList name="regexp-list">
                            <string>.*core\.UserAdmin.*core\.UserLogin.*</string>
                            <string>.*core\.UserAdmin.*register\.UserSelfRegistration.*</string>
                            <string>.*\/w\/index\.php\?title=Speci[ae]l:Recentchanges.*</string>
                            <string>.*act=calendar&amp;cal_id=.*</string>
                            .....
                            <string>.*calendar\.asp\?qMonth=.*</string>
                            <string>.*calendar\.php\?sid=.*</string>
                            <string>.*worldscinet\.com.*</string>
                            <string>.*www3\.interscience\.wiley\.com.*</string>
                            <string>.*www-gdz\.sub\.uni-goettingen\.de.*</string>
                        </stringList>
                    </newObject>
                </map>
            </newObject>
```

3) Additional filters. Here we have a "Force-accept-filter", an "additionalScopeFocus" filter, and a "transitive
Filter", of which only the transitiveFilter element needs to be converted. The two other elements are just
deleted.

```
            <newObject name="force-accept-filter" class="org.archive.crawler.filter.OrFilter">
                <boolean name="enabled">true</boolean>
                <boolean name="if-matches-return">true</boolean>
                <map name="filters">
                </map>
            </newObject>
            <newObject name="additionalScopeFocus" class=
"org.archive.crawler.filter.FilePatternFilter">
                <boolean name="enabled">true</boolean>
                <boolean name="if-match-return">true</boolean>
                <string name="use-default-patterns">All</string>
                <string name="regexp"/>
            </newObject>
            <newObject name="transitiveFilter" class="org.archive.crawler.filter.TransclusionFilter">
                <boolean name="enabled">true</boolean>
                <integer name="max-speculative-hops">1</integer>
                <integer name="max-referral-hops">15</integer>
                <integer name="max-embed-hops">15</integer>
            </newObject>
        </newObject> <!-- end of scope element -->
```

== How to convert from the former scopes to a decidingscope ==

Converting the header is easy.
All headers have the form:

```
  <newObject name="scope" class="org.archive.crawler.deciderules.DecidingScope">
            <boolean name="enabled">true</boolean>
            <string name="seedsfile">seeds.txt</string>
            <boolean name="reread-seeds-on-config">true</boolean>
            <!-- DecideRuleSequence. Multiple DecideRules applied in order with last non-PASS the
resulting decision -->
            <newObject name="decide-rules" class="org.archive.crawler.deciderules.DecideRuleSequence">

        <map name="rules">
    <newObject name="rejectByDefault"
     class="org.archive.crawler.deciderules.RejectDecideRule"/>
```

plus a special defining deciderule that emulates the DomainScope, the HostScope, or the PathScope.
1) The defining deciderule for DomainScope is (the only one using a special purpose DecideRule):

```
<newObject name="acceptURIFromSeedDomains" class=
"dk.netarkivet.harvester.harvesting.OnNSDomainsDecideRule">
                            <string name="decision">ACCEPT</string>
                            <string name="surts-source-file">seeds.txt</string>
                            <boolean name="seeds-as-surt-prefixes">false</boolean>
                            <string name="surts-dump-file"/>
                            <boolean name="also-check-via">false</boolean>
                            <boolean name="rebuild-on-reconfig">true</boolean>
</newObject>
```

2) The defining deciderule for HostScope is:

```
<newObject name="OnHostsRule" class="org.archive.crawler.deciderules.OnHostsDecideRule">
          <string name="decision">ACCEPT</string>
          <string name="surts-dump-file"/>
          <boolean name="also-check-via">false</boolean>
          <boolean name="rebuild-on-reconfig">true</boolean>
      </newObject>
```

3) The defining deciderule for PathScope is:

```
<newObject name="acceptIfSurtPrefixed" class="org.archive.crawler.deciderules.SurtPrefixedDecideRule">
                        <string name="decision">ACCEPT</string>
                        <string name="surts-source-file"></string>
                        <boolean name="seeds-as-surt-prefixes">true</boolean>
                        <string name="surts-dump-file"></string>
                        <boolean name="also-check-via">false</boolean>
                        <boolean name="rebuild-on-reconfig">true</boolean>
</newObject>
```

After the header and the defining deciderule, we a deciderule corresponding to the 'hops_filter'.
Note that the two last attributes 'max-link-hops', and 'max-trans-hops' in the header cease to be general
scope attributes. Instead max-trans-hops become an attribute for the "acceptIfTranscluded" mentioned above,
and the 'max-link-hops' attribute becomes an attribute for the new 'hops_filter' deciderule.
The following

```
<integer name="max-link-hops">10</integer>
<newObject name="hops_filter" class="org.archive.crawler.filter.HopsFilter">
 <boolean name="enabled">true</boolean>
</newObject>
```

is then translated to the following deciderule

```
<newObject name="rejectIfTooManyHops" class="org.archive.crawler.deciderules.TooManyHopsDecideRule">
 <integer name="max-hops">10</integer>
</newObject>
```

Following this, we need to add a translation of the 'pathdepth' element, and the 'pathologicalpath'
element, plus a translation of the 'transitiveFilter' element in the last part of the scope.
The following

```
            <newObject name="pathdepth" class="org.archive.crawler.filter.PathDepthFilter">
                            <boolean name="enabled">true</boolean>
                            <integer name="max-path-depth">20</integer>
                            <boolean name="path-less-or-equal-return">false</boolean>
            </newObject>
            <newObject name="pathologicalpath" class="org.archive.crawler.filter.PathologicalPathFilter">
               <boolean name="enabled">true</boolean>
                <integer name="repetitions">3</integer>
            </newObject>

            <newObject name="transitiveFilter" class="org.archive.crawler.filter.TransclusionFilter">
                      <boolean name="enabled">true</boolean>
                      <integer name="max-speculative-hops">1</integer>
                      <integer name="max-referral-hops">15</integer>
                      <integer name="max-embed-hops">15</integer>
            </newObject>
```

is translated to

```
<newObject name="rejectIfPathological" class=
"org.archive.crawler.deciderules.PathologicalPathDecideRule">
 <integer name="max-repetitions">3</integer>
</newObject>
<newObject name="acceptIfTranscluded" class="org.archive.crawler.deciderules.TransclusionDecideRule">
 <integer name="max-trans-hops">5</integer>
 <integer name="max-speculative-hops">1</integer>
</newObject>
<newObject name="pathdepthfilter" class=
"org.archive.crawler.deciderules.TooManyPathSegmentsDecideRule">
 <integer name="max-path-depth">20</integer>
</newObject>
```

.

Note that the attributes 'max-referral-hops' and 'max-embed-hops' in the 'transitiveFilter' element
have been merged into one single attribute 'max-trans-hops' which is now no longer an attribute
of the scope, as it was in the old scopes.

Now you only need to convert all remaining URIRegExpFilter and URIListRegExpFilter elements to a
corresponding DecideRule. The deciderule corresponding to URIRegExpFilter is MatchesRegExpDecideRule, and
the deciderule corresponding to URIListRegExpFilter is MatchesListRegExpDecideRule.
Converting the dr_dk element (a URIRegExpFilter)

```
    <newObject name="dr_dk" class="org.archive.crawler.filter.URIRegExpFilter">
                            <boolean name="enabled">true</boolean>
                            <boolean name="if-match-return">true</boolean>
                            <string name="regexp">.*dr\.dk.*epg\.asp.*</string>
                    </newObject>
```

gives us:

```
  <newObject name="dr_dk" class="org.archive.crawler.deciderules.MatchesRegExpDecideRule">
        <string name="decision">REJECT</string>
        <string name="regexp">.*dr\.dk.*epg\.asp.*</string>
   </newObject>
```

Converting the globale_crawlertraps element (URIListRegExpFilter)

```
<newObject name="globale_crawlertraps" class="org.archive.crawler.filter.URIListRegExpFilter">
                    <boolean name="enabled">true</boolean>
                    <boolean name="if-match-return">true</boolean>
                    <string name="list-logic">OR</string>
                    <stringList name="regexp-list">
                        <string>.*core\.UserAdmin.*core\.UserLogin.*</string>
                        <string>.*core\.UserAdmin.*register\.UserSelfRegistration.*</string>
                        <string>.*\/w\/index\.php\?title=Speci[ae]l:Recentchanges.*</string>
                        <string>.*act=calendar&amp;cal_id=.*</string>
                        .....
                        <string>.*calendar\.asp\?qMonth=.*</string>
                        <string>.*calendar\.php\?sid=.*</string>
                        <string>.*worldscinet\.com.*</string>
                        <string>.*www3\.interscience\.wiley\.com.*</string>
                        <string>.*www-gdz\.sub\.uni-goettingen\.de.*</string>
                    </stringList>
                </newObject>
```

gives us

```
<newObject name="globale_crawlertraps" class=
"org.archive.crawler.deciderules.MatchesListRegExpDecideRule">
            <string name="decision">REJECT</string>
            <string name="list-logic">OR</string>
            <stringList name="regexp-list">
        <string>.*core\.UserAdmin.*core\.UserLogin.*</string>
                <string>.*core\.UserAdmin.*register\.UserSelfRegistration.*</string>
                <string>.*\/w\/index\.php\?title=Speci[ae]l:Recentchanges.*</string>
                <string>.*act=calendar&amp;cal_id=.*</string>
                .....
                <string>.*calendar\.asp\?qMonth=.*</string>
                <string>.*calendar\.php\?sid=.*</string>
                <string>.*worldscinet\.com.*</string>
                <string>.*www3\.interscience\.wiley\.com.*</string>
            </stringList>
</newObject>
```

Finally we need to wrap up the the sequence of deciderules and the scope itself.
So we add

```
   </map> <!-- end rules -->
            </newObject> <!-- end decide-rules  -->
        </newObject> <!-- End DecidingScope -->
```

.

Appendix_B