Detailed Configurations

Contents

Configurations Affecting JMS Channel Names

JMS 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 is more than one instance of the same application running on the same machine, e. g. when several 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 distinct 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.

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.

Configuring Plugins

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 specific implementation.

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

For more details on how to extend the system with pluggable classes with their own settings, please see the System Design manual 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). These are examples of settings which are absent from the default settings files distributed with NetarchiveSuite, because they are only relevant for an optional plugin.:

```
<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* the 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 an 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 are only 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.

The NetarchiveSuite currently only supports one kind of JMS broker, so only the 'broker' and 'port' 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

NetarchiveSuite archives its harvested data in a component called an ArcRepository. The name is a historical relic as the current version supports archiving of both arcfiles and warcfiles.

The repository is configured either as a simple local arcrepository or a complex distributed arcrepository having multiple 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). In this case the data is simply stored on one or more local mounts on the machine where the `ArcRepositoryApplication` is running.

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 need 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 usage of Bitrepository/Hadoop backend

With the release of NetarchiveSuite 7.0, it is possible to switch the built-in repository/bitarchive setup with the one provided by the Bitrepository /BITMAG software, which provides a more extensive architecture for the distributed preservation of data. Note that the Bitrepository software is not a built-in part of NetarchiveSuite so this must be set up externally (details on how to do this can be on the provided link).

Furthermore, with this setup most batch jobs concerned with e.g. CDX-indexing and metadata extraction are expected to be handled by an external Hadoop cluster as MapReduce jobs. Local Hadoop settings must be set up correctly to point to this cluster on the machines running WaybackIndexer-, IndexServer-, and ViewerProxyApplication. Kerberos is used for secure communication with the cluster, which must also be set up for the cluster (see below Hadoop settings for Kerberos configurations within NetarchiveSuite).

To enable the Bitrepository/Hadoop backend `settings.common.useBitmagHadoopBackend` must be set to true. As the ArcRepository (described in the above section) uses the Bitmag software `settings.common.arcrepositoryClient.class` must also be configured to use `dk.netarkivet.archive.arcrepository.distribute.BitmagArcRepositoryClient`.

Configuring Bitrepository settings

Below are the settings listed that are used to configure the Bitrepository software's use.

settings.common.arcrepositoryClient.bitrepository.settingsDir: Path to the settings directory in which files like Repository-/ReferenceSettings.xml reside.

settings.common.arcrepositoryClient.bitrepository.keyfilename: Name of the keyfile in the above settings dir to use for secure communication with Bitrepository. If not set or the name does not exist, use of certificates is disabled.

settings.common.arcrepositoryClient.bitrepository.tempdir: Setting for where the bitrepository has its temporary directory. (currently unused?)

settings.common.arcrepositoryClient.bitrepository.collectionID: ID of the bitrepository collection to perform operations on. If not set, the environment name set by `settings.common.environmentName` is used instead.

settings.common.arcrepositoryClient.bitrepository.usepillar: Name/ID of the pillar to use when getting file IDs that are in the repository.

settings.common.arcrepositoryClient.bitrepository.getFileIDsMaxResults: Setting specifying the max amount of IDs to fetch when making a GetFileIDs operation.

settings.common.arcrepositoryClient.bitrepository.getTimeout: Setting for how many milliseconds we will wait before giving up on a get request to the Arcrepository. (currently unused?)

settings.common.arcrepositoryClient.bitrepository.storeMaxPillarFailures: Setting for how many pillars are allowed to fail on a store/putFile operation.

settings.common.arcrepositoryClient.bitrepository.storeRetries: How many times to retry a store/put operation before giving up and throwing an error.

settings.common.arcrepositoryClient.bitrepository.retryWaitSeconds: How long to wait between retries of a store operation.

Configuring Hadoop settings

In order for communication with the Hadoop cluster to work, the following security settings for Kerberos must first be configured:

settings.common.hadoop.kerberos.principal: The Kerberos principal (username) to use when running jobs.

settings.common.hadoop.kerberos.keytab: Path to the keytab to use for authentication.

settings.common.hadoop.kerberos.krb5-conf: Path to the Kerberos configuration file to use when running jobs. Kerberos uses /etc/krb5.conf by default.

The rest of the Hadoop configurations are as shown below:

settings.common.hadoop.mapred.hadoopUberJar: Path to shaded "uber" jar containing all dependencies that the MapReduce jobs require at runtime. Hadoop nodes running the jobs will fail at runtime if the jar is incomplete.

settings.common.hadoop.mapred.inputFilesParentDir: Path to the shared directory between the Hadoop nodes in which the input files for MapReduce jobs can be found.

settings.common.hadoop.mapred.cdxJob.inputDir/.outputDir: Name of the input/output directory on the Hadoop File System (HDFS) for CDX-indexing jobs.

settings.common.hadoop.mapred.metadataExtractionJob.inputDir/.outputDir: Name of the input/output directory on HDFS for metadata extraction jobs.

settings.common.hadoop.mapred.metadataCDXExtractionJob.inputDir/.outputDir: Name of the input/output directory on HDFS for jobs extracting CDX lines from metadata files.

settings.common.hadoop.mapred.crawlLogExtractionJob.inputDir/.outputDir: Name of the input/output directory on HDFS for jobs extracting crawl log lines from metadata files.

Configure job-generation

The scheduling of new harvest jobs takes place every one minute, unless the previous scheduling is not finished yet. The scheduling interval can be changed by altering the setting

settings.harvester.scheduler.jobgenerationperiod: In seconds, default value 60

Scheduling consists of searching for active harvestdefinitions that are ready to have jobs generated, and subsequently submitted, for harvesting. The job-generation procedure is 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 they may take to complete. Note that a harvestdefinition 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 particularly slow or large domains causing harvests to take longer than necessary. In the job splitting part of the scheduling, the scheduler partitions a (possibly) large number of DomainConfigurations into several crawljobs. Each crawljob can have only one Heritrix setup (crawler-bean definition), 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 crawler-bean templates, or if different harvest limits need to be enforced.

settings.harvester.scheduler.errorFactorPrevResult: Used when calculating the 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 at most 10 times as great as the value we use as the expected size.

settings.harvester.scheduler.errorFactorBestGuess: Used when calculating the expected size of a harvest of some domain during the job-creation process for a snapshot harvest. The default value is 2.

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 the web develops. Default is 38000.

settings.harvester.scheduler.maxDomainSize: Initial guess of number 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. Default values is 100.

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 8000000 objects. (This value was raised in NetarchiveSuite 5.4 from the previous value of 2000000 which was found to generate too many jobs in a snapshot harvest of the .dk domain.)

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. From NetarchiveSuite 5.2 and up, the recognised list of TLDs is taken directly from ICANN, and downloaded from https://www.publicsuffix.org/list/public_suffix_list.dat . This list can be overridden if necessary by placing a new list at the hard-coded path conf/public_suffix_list.dat in the installation on the machine where the GUIApplication and HarvestJobManager run.

In addition, the setting `settings.common.topLevelDomains.tld` can be used to define additional allowed TLDs.

Configure Heritrix process

In this section the configuration for running Heritrix processes via NetarchiveSuite is described. For details on managing heritrix harvest templates (.cxml files), please refer to Appendix B2: Managing Heritrix 3 Crawler-Beans.

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 dk.netarkivet.harvester.heritrix3.controller.HeritrixController.

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. In NetarchiveSuite 5.2 and up, this functionality is now also available directly from within the NetarchiveSuite GUI. Note that some changes to harvests, especially those that change the scope and limits, may confuse the harvest definition system. It is safest to use 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. https://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. These have default values "admin" and "adminPassword".

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 (see template in examples/jmxremote_template.password).

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>
```

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 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 and behaviour

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 suit 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 customised if you like. From NetarchiveSuite 5.3, these logo files can also be replaced by custom logos using the deploy application.

In addition, the setting `settings.harvester.webinterface.maxCrawlLogInBrowser` controls the display of crawl logs when browsing the output of a harvest job. If the crawl log selection is longer than this parameter then the result is saved to a file rather than being displayed in the browser window. The default value is 100.

The setting `settings.harvester.webinterface.hideInactiveTemplates` determines what functionality is available in the **Edit Harvest Templates** page of the GUI. The default value is false, in which case users can move templates between Active and Inactive states. If set to true, this functionality is not available. (See the User Manual for screenshots of the effect of changing this parameter.)

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 vcs repository, as we do at the Netarkivet. In Netarkivet, the java security policy is only applied to instances of `dk.netarkivet.archive.bitarchive.BitarchiveApplication` as these are the only applications which have direct access to the archive files and on which it is in principle possible to run 3rd party code.

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.netarkivetorg.netarchivesuite.* 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 use 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 flat files or a database. The default behaviour is to use flat 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>
```

Here is an example of a database setup using postgres:

```
<settings>
       <archive>
         <admin>
           <class>dk.netarkivet.archive.arcrepositoryadmin.DatabaseAdmin</class>
           <database>
             <class>dk.netarkivet.archive.arcrepositoryadmin.PostgreSQLSpecifics</class>
             <baseUrl>jdbc:postgresql</baseUrl>
             <machine>localhost</machine>
             <port>5432</port>
             <dir>admindb</dir>
             <username>devel</username>
             <password>develpass</password>
             <pool>
               <idleConnTestQuery>SELECT COUNT(*) FROM replica</idleConnTestQuery>
               <idleConnTestOnCheckin>false</idleConnTestOnCheckin>
             </pool>
           </database>
         </admin>
       </archive>
       <common>
         <database>
           <class>dk.netarkivet.harvester.datamodel.PostgreSQLSpecifics</class>
           <baseUrl>jdbc:postgresql</baseUrl>
           <machine>localhost</machine>
           <port>5432</port>
           <dir>harvestdb</dir>
           <username>devel</username>
           <password>develpass</password>
         </database>
       </common>
     </settings>
```

Examples of deploy configuration files

The Quickstart configuration file requires adaptation to your own system before use.