

Appendix A - Plug-ins in NetarchiveSuite

Contents

All NetarchiveSuite settings ending in ".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. Our framework enables the installer to replace the default class with a class of their own, if no existing alternatives are suitable.

Setting (dk.netarkivet:)	Interface or Abstract Class (dk.netarkivet:)	Concrete Implementations Provided - ★ = default (dk.netarkivet:)
.common.remoteFile.class	.common.distribute. AbstractRemoteFile	.common.distribute.HTTPRemoteFile .common.distribute.HTTPSRemoteFile .common.distribute.FTPRemoteFile ★
.archive.bitpreservation. class	.archive.arcrepository. bitpreservation. ActiveBitPreservation	.archive.arcrepository.bitpreservation. DatabaseBasedActiveBitPreservation .archive.arcrepository.bitpreservation. FileBasedActiveBitPreservation ★
.archive.admin.class	.archive.arcrepositoryadmin.Admin	.archive.arcrepositoryadmin. UpdateableAdminData .archive.arcrepositoryadmin.DatabaseAdmin ★
.archive.admin.database. class:	.archive.arcrepositoryadmin. DBSpecifics	.archive.arcrepositoryadmin. DerbyServerSpecifics ★ .archive.arcrepositoryadmin. DerbyEmbeddedSpecifics .archive.arcrepositoryadmin.MySQLSpecifics .archive.arcrepositoryadmin. PostgreSQLSpecifics
.archive.checksum.archive. class	.archive.checksum.ChecksumArchive	.archive.checksum.FileChecksumArchive ★ .archive.checksum.DatabaseChecksumArchive
.common.jms.class	.common.distribute.JMSConnection	.common.distribute.JMSConnectionSunMQ ★
.common.arcrepositoryClient. class	.common.distribute. ArcRepositoryClient	.archive.arcrepository.distribute. JMSArcRepositoryClient ★ .archive.arcrepository.distribute. BitmagArcRepositoryClient .common.distribute. LocalArcRepositoryClient
.common. monitorregistryClient.class.	.common.distribute. monitorregistry. MonitorRegistryClient	.common.distribute.monitorregistry. PrintMonitorRegistryClient .monitor.distribute. JMSMonitorRegistryClient ★
.common.indexClient.class	.common.distribute.indexserver. JobIndexCache	.harvester.indexserver.distribute. IndexRequestClient ★
.common.notifications.class	.common.utils.Notifications	.common.utils.EMailNotifications ★ .common.utils.PrintNotifications

<code>.common.freespaceprovider.class</code>	<code>.common.utils.FreeSpaceProvider</code>	<code>.common.utils.DefaultFreeSpaceProvider</code> ★ <code>.common.utils.Filebased.FreeSpaceProvider</code> <code>.common.utils.OnbFreeSpaceProvider</code>
<code>.common.fileResolver.class</code>	<code>.common.utils.service.FileResolver</code>	<code>.common.utils.service.FileResolverRESTClient</code> ★ <code>.common.utils.service.SimpleFileResolver</code>
<code>.common.database.class</code>	<code>.harvester.datamodel.DBSpecifics</code>	<code>.harvester.datamodel.DerbyServerSpecifics</code> ★ <code>.harvester.datamodel.DerbyEmbeddedSpecifics</code> <code>.harvester.datamodel.MySQLSpecifics</code> <code>.harvester.datamodel.PostgreSQLSpecifics</code>
<code>.harvester.scheduler.jobGen.class</code>	<code>.harvester.scheduler.jobgen.JobGenerator</code>	<code>.harvester.scheduler.jobgen.DefaultJobGenerator</code> ★ <code>.harvester.scheduler.jobgen.FixedDomainConfigurationCountJobGenerator</code>
<code>.harvester.harvesting.heritrix.archiveNaming.class</code>	<code>.harvester.harvesting.ArchiveFileNaming</code>	<code>.harvester.harvesting.LegacyNamingConvention</code> ★ <code>.harvester.harvesting.CollectionPrefixNamingConvention</code>
<code>.frontier.filter.class</code>	<code>.harvester.harvesting.frontier.FrontierReportFilter</code>	<code>.harvester.harvesting.frontier.TopTotalEnqueuesFilter</code> ★ <code>.harvester.harvesting.frontier.ExhaustedQueuesFilter</code> <code>.harvester.harvesting.frontier.MaxSizeFrontierReportExtract</code> <code>.harvester.harvesting.frontier.RetiredQueuesFilter</code>
<code>.heritrixLauncher.class</code>	<code>.harvester.heritrix3.HeritrixLauncherAbstract</code>	<code>.harvester.heritrix3.controller.HeritrixLauncher</code> ★
<code>.heritrixController.class</code>	<code>.harvester.heritrix3.controller.IHeritrixController</code>	<code>.harvester.heritrix3.controller.HeritrixController</code> ★
<code>.harvestReport.class</code>	<code>.harvester.harvesting.report.HarvestReport</code>	<code>.harvester.harvesting.report.LegacyHarvestReport</code> ★ <code>.harvester.harvesting.report.BnFHarvestReport</code>
<code>.indexrequestserver.class</code>	<code>.harvester.indexserver.IndexRequestServerInterface</code>	<code>.harvester.indexserver.distribute.IndexRequestServer</code> ★

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

settings.common.remoteFile.class: This setting allows you to select your chosen protocol for file-transfer in NetarchiveSuite. You can here choose between

- `dk.netarkivet.common.distribute.FTPRemoteFile`: (where the data is transferred using a FTP-server),
- `dk.netarkivet.common.distribute.HTTPRemoteFile`: (where the data is transferred using two embedded webservers (one at each end)), and

- `dk.netarkivet.common.distribute.HTTPSRemoteFile`: which works just like `HTTPRemoteFile` except it uses a shared certificate file for secure communication. Note that the `HTTPRemoteFile` and `HTTPSRemoteFile` require dedicated ports in the firewall to be open between all possible senders and recipients of data. For implementers of new file-transfer methods, this class must implement the interface `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. The supported values are:

- `dk.netarkivet.harvester.datamodel.DerbyEmbeddedSpecifics`: an embedded derby database
- `dk.netarkivet.harvester.datamodel.DerbyClientSpecifics`: an external derby database server
- `dk.netarkivet.harvester.datamodel.MySQLSpecifics`: `mySql`
- `dk.netarkivet.harvester.datamodel.PostgreSQLSpecifics`: `postgresql`

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`). The Netarkivet production system uses `Postgresql`.

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 `dk.netarkivet.common.distribute.JMSConnection`.

settings.common.arcrepositoryClient.class. Must implement `dk.netarkivet.common.distribute.ArcRepositoryClient` The available choices are:

- `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). This is the default class.
- `dk.netarkivet.archive.arcrepository.distribute.BitmagArcRepositoryClient` (allows use of [the Bitrepository software](#) as archive storage by sending requests directly to its messagebus. Requires the Bitrepository/Hadoop backend enabled by having the setting `settings.common.useBitmagHadoopBackend` set to `true`)
- `dk.netarkivet.common.distribute.LocalArcRepositoryClient` (allows for access to a local archive, ie a collection of files on locally-mounted filesystems.)

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 6 existing webmodules:

1. `HarvestDefinition`: Allows you to define and schedule harvests ,
2. `HarvestHistory`: See the status of running and finished harvestjobs
3. `HarvestChannel`: Allows one to define different channels for different classes of so that, for example, specific harvest jobs can be run on specific machines.
4. `BitPreservation`: This module has tools for sanity testing data in the bitarchives
5. `QA`: Module for doing Quality Assurance
6. `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`), French (`locale=fr`), 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 `HarvesterControllerApplication` 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 the monitor registry, which implement the interface `dk.netarkivet.common.distribute.monitorregistry.MonitorRegistryClient`. There are 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 setting `settings.common.monitorregistryClient.reregisterdelay`.

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.common.fileResolver.class*: Setting defining which class to use for resolving file paths when using Hadoop for mass processing. All implementations must implement `dk.netarkivet.common.utils.service.FileResolver`. Available implementations are:

- `dk.netarkivet.common.utils.service.FileResolverRESTClient` (resolves file paths by sending requests to a REST-based service located at the URL specified by `settings.common.fileResolver.baseUrl`)
- `dk.netarkivet.common.utils.service.SimpleFileResolver` (resolves file paths by looking in a specified local parent directory.)

The default class is `dk.netarkivet.common.utils.service.FileResolverRESTClient`.

*The setting is only relevant/in use when the Bitrepository/Hadoop backend is enabled i.e. `settings.common.useBitmagHadoopBackend` is set to true.

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 an 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 `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. In NAS 5.2+, the only supported heritrix controller is `dk.netarkivet.harvester.heritrix3.controller.HeritrixController`.

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.

