
# kopal Library for Retrieval and Ingest

– DOCUMENTATION –

Stefan Funk, Kadir Karaca Koçer, Sabine Liess, Jens Ludwig, Matthias Neubauer

© Project kopal,

German National Library /

Goettingen State and University Library

koLibRI v1.0 – July 2007

# Contents

# 1 kopal Library of Retrieval and Ingest – an overview

*koLibRI* is a framework for the integration of a long-term archiving system, like the IBM Digital Information Archiving System [1] (DIAS), into the infrastructure of an institution. In particular, it organizes the creation and ingest of archival packages into DIAS and provides functionalities to retrieve and manage these packages.

This document describes the installation and configuration of a fully functional koLibRI system, as well as its basic internal design to allow individual developments and extensions. A certain amount of basics about long-term archiving [7], as well as about the standards OAIS [2], URN [3], METS [4], LMER [5] und IBM DIAS is essential for fully understanding the system. The necessary knowledge is, however, freely available via the Internet.

koLibRI is being provided in full functionality and a stable state after the succesful completion of the kopal project [6]. The terms of use and liability apply as stated on the kopal website[1].

## 1.1 Functionality

In short, koLibRI generates a XML file according to the METS schema out of the metadata, provided with the object to archive or generated by JHOVE [8], bundles it into an archive file together with the object (.zip or .tar) and delivers this *Submission Archiving Package* (SIP) to the DIAS system.

From that point of view, koLibRI was developed to provide a full implemented long-term archiving solution together with the IBM DIAS. However, koLibRI can also be used as an independent software to create METS files or whole SIPs according to the Universal Object Format [22], completely without the need of the DIAS system. XML metadata files or SIPs generated in this way, can be used for data exchange between several institutions; a feature which was one of the leading aspects in the development of the UOF. Because of its modular design and its open specified interfaces, koLibRI can alternatively be adapted to another archival system or metadata format with affordable effort.

### 1.1.1 Ingest

It has to be defined which working steps are necessary to create and ingest an archival package. These steps can differ for different types of digital objects.

[1] http://kopal.langzeitarchivierung.de/index_koLibRI.php.en

For instance, the sources of objects (CD-ROM within a local machine, document server in the intranet, etc.) and their descriptive metadata (XML file attached to the object, OAI interface of a catalog system, etc.) strongly vary for electronic theses, digitized books and even within these object classes. koLibRI was designed modular, so that these different steps can be integrated. It allows the notation of these steps within a XML file called `policies.xml`. The working steps are referred to as `steps`, which are implemented by `action modules`.

Typical modules process tasks like the download of files for the archival package, the validation of the files, extraction of technical metadata, generation of the metadata file that belongs to the UOF and the ingest of the final archival package into DIAS. There is also the possibility to scale these steps to different machines, as to create archival packages within several departments, but to use a central instance for quality assurance, entries into catalog systems and the final ingest.

### 1.1.2 koLibRI-Database

For many user scenarios, it is useful to harvest and store data about processed objects during the ingest process. To do this, koLibRI provides database support, so that own identifiers, Dublin Core metadata or some technical metadata for instance can be stored file based. This way, the data is available also for functionalities outside of koLibRI, e.g. for statistics, identifier resolving or an OAI interface.

### 1.1.3 Retrieval

The functionality of retrieving archive objects is usable via the Web Service (see chapter 9 on Web Service). On the other hand, one can use the command line tool `DiasAccess`, and the according Java methods of the same class within own software respectively.

## 1.2 Project scale

koLibRI has been developed within the scale of project kopal – Co-operative Development of a Long-Term Digital Information Archive [6] by the German National Library and the Goettingen State and University Library for the use of the system `kopal solution`. Subject of the 3-year project kopal (2004 - 2007) was the practical proving and implementation of a cooperative developed and operated long-term archiving system for digital publications.

As partner, the German National Library [18], the Goettingen State and University Library [19] and IBM Germany [20] have implemented a cooperative and reusable solution

for the long-term preservation of digital resources. The technical maintenance is done by the Gesellschaft fuer wissenschaftliche Datenverarbeitung mbH Goettingen (GWDG) [21].

## 1.3 Distribution

The precompiled binaries and the source code of koLibRI can be downloaded from

http://kopal.langzeitarchivierung.de/kolibri

free of charge or registration. Please send an email to

kolibri@kopal.langzeitarchivierung.de

when facing any problems.

Everyone interested in the software is free to modify or extend it and can pass it on to others. However, the licence agreements that are being provided by kopal have to be acknowledged. The software libraries and service programs that have been developed of the kopal project, and are only being used by koLibRI, partially apply to other licences which are available in the corresponding license files. Any questions regarding these software libraries are only to be addressed to the respective authors of the according software package.

# 2 Installation and configuration

## 2.1 Requirements

koLibRI has been implemented in Java only, that means it should be usable on any platform that provides a Java Virtual Machine in version 1.5 (However, even alternating Java environments are not always compatible. Problems with alternating XML parsers where sporadically observed). The development team has tested koLibRI on various processor architectures and operating systems[2].

koLibRI is free software in the sense of the Free Software Foundation [9] It can likely be ported to other platforms in favour. In such cases, the kopal team would be pleased to be informed about any kind of customization.

## 2.2 Installation

The installation of the precompiled distribution consists of the following components:

1. The precompiled program package `kolibri.jar`,
2. the directory with linked software packages `/lib`, and
3. the example and main configuration files in the `/config` directory.

In addition there are several batch files for MS Windows and Unix to provide an easy way of the invocation of the included software tools. These components simply have to be unpacked from the packed archive file into a favoured directory. If one wishes to recompile koLibRI from the source files by himself, the packed `.java` files simply have to be unpacked and compiled – preferential with the included Ant script.

After `kolibri.jar` is available on the hard disk, the batch file `workflowtool.bat` (for MS Windows), respectively its pendant for Unix, has to be customized with the help of a text editor. Once all paths and parameters have been set correctly, the different functionalities of the program are available by the invocation of the according batch files.

The important thing at this point is to know, which paths and parameters are appropriate for the according system. In cases of uncertainness, the according system administrator should be contacted. It is recommended – if available – to reserve 512 megabytes or more of

[2] different Linux distributions, Mac OS X 10.4 and MS Windows XP

```
#!/bin/sh

##############################################################################
# Copyright 2005-2007 by Project kopal
#
# http://kopal.langzeitarchivierung.de/
#

 ...
 ...

# Configuration constants:

KOLIBRI_HOME=/users/user/projects/kopal/kolibri  # The koLibRI location
JAVA_HOME=/usr/java                              # Java JRE directory
JAVA=$JAVA_HOME/bin/java                         # Java interpreter
EXTRA_JARS=lib/clibwrapper_jiio.jar: ... ...

# NOTE: Nothing below this line should be edited
##############################################################################

CP=${KOLIBRI_HOME}/kolibri.jar:${EXTRA_JARS}

# Retrieve a copy of all command line arguments to pass to the application.

ARGS=""
for ARG do
    ARGS="$ARGS $ARG"
done

# Set the CLASSPATH and invoke the Java loader.
${JAVA} -classpath $CP -Xmx512m de.langzeitarchivierung.kopal.WorkflowTool $ARGS

##############################################################################
```

Figure 1: The file `workflowtool`

heap size through the parameters.

Listed below are the main parts of the workflowtool.bat file. They are followed by comments to the lines that have to be customized:

- `KOLIBRI_HOME` – the directory in which the kolibri.jar file is located.
- `JAVA_HOME` – the directory of the local Java installation.
- `JAVA` – the directory within `JAVA_HOME`, which contains the executable Java Virtual Machine.
- `EXTRA_JARS` – complete paths and filenames to optional software libraries to be linked,

which are to be separated by a semicolon (MS Windows) or a colon (Unix). In example, this can be institution specific software extensions for koLibRI.

## 2.3 Configuration of koLibRI

To configure koLibRI, the files `config.xml` (an alternative file name or path can be used through the command line parameter "-c") and the policy file `policies.xml` have to be customized. The config file contains global and module/class specific configuration parameters. Configuration values for individual working steps can be set within the policy file.

The most specific value has the highest priority, that means the Configurator first tries to fnd a parameter for the actual step, then for the actual module or class and finally it uses global configuration parameters.

The configuration file has a structure as follows (for the formal specification please see the file `config.xsd`):

```
<config>
  <common>
    <property>
      <field>logLevel</field>
      <value>INFO</value>
      <description>...</description>
    </property>
  </common>

  <modules>
    <class name="actionmodule.MetadataGenerator">
      <property>
        <field>acceptedUnknownFileFormats</field>
        <value>.xms</value>
        <value>.doc</value>
        <description>...</description>
      </property>
    </class>
  </modules>
</config>
```

The `common` block defines the global parameters, the `modules` block the module and class specific ones. Each configuration parameter is embedded within `property` tags. The `field` tag defines the name of the parameter (case sensitivity is not relevant) and the according

values are set within one ore more `value` tags. The repeatability of a `value` tag is dependent of the according parameter, whose meaning is documented within the `description` tags. The `class` tag bundles all configuration parameters for a specific module, where the `name` attribute is stating the name of the according Java module (either by the full qualified package name or by abandonment of the usual prefix `de.langzeitarchivierung.kopal`).

Setting configuration values for individual working steps of the policy file is done in the following way (for the formal specification please see the file `policies.xsd`, for a description of the policy file please see section 3 on page 11):

```
<step class="MetadataGenerator">
  <config>
    <property>
      <field>showPdfPages</field>
      <value>false</value>
    </property>
  </config>
  ...
</step>
```

If a config tag is present one hierarchical level below a `step` tag, all configuration parameters contained in the `config` tag are used for the specific class of the `step` tag, for documentation please see example configuration files. It is suggested to use the provided examples as a basis for individual configurations.

# 3 Using koLibRI for the ingest workflow

To use koLibRI for the ingest of archival packages it is essential to specify the working steps in the configuration file `policies.xml`[3]. The formal syntax is defined in the example file `policies.xsd` within the config directory, the functionality of the example workflows, together with the example configuration files, is explained in section 3.2 on page 19.

Workflows are seen as trees in the sense of the graph theory, that is processed in the direction from the root to the leafs. Each node is a step, whose child nodes are only processed, if the parent node was processed successfully. The child nodes are processed parallel if applicable, this can be very useful for time consuming tasks (like burning a CD-ROM or the transfer to another archive)[4].

An example workflow would process the following six steps in a row:

1. Copy selective files to a process directory
   (ActionModule `FileCopyBase`),
2. extract descriptive metadata for the files
   (ActionModule `MetadataExtractorDmd`),
3. generate technical metadata for the files (ActionModule `MetadataGenerator`, please see section 10.1 on page 51),
4. generate metadatafile `mets.xml`
   (ActionModule `MetsBuilder`),
5. compress all files into an archival package
   (ActionModule `Zip`), and finally
6. ingest the archival package into DIAS
   (or another archive system)
   (ActionModule `SubmitDummySipToArchive`).

Please have a look at the related policy file:

```
<policies>
  <policy name="example">
    <step class="FileCopyBase">
```

[3] Which steps are needed is also dependent of the individual archiving requirements. Each institution has to declare these requirements for itself; the process of finding a long-term archiving policy can not be replaced by a technical solution (please see partner project nestor [7] for further information).

[4] This functionality is not yet realized

```
      <step class="MetadataExtractorDmd">
        <step class="MetadataGenerator">
          <config>
            <property>
              <field>showHtmlImages</field>
              <value>true</value>
              <description>...</description>
            </property>
          </config>
          <step class="MetsBuilder">
            <step class="Zip">
              <step class="SubmitDummySipToArchive">
              </step>
            </step>
          </step>
        </step>
      </step>
    </step>
  </policy>
</policies>
```

Besides the working steps itself, it has to be defined, how new working steps are being started and which initialization values they get. This is the meaning of the so called `ProcessStarters`, which are set at the start of the program either through the command line parameter `-p`, or in the configuration file through the parameter `defaultProcessStarter`. For basic usage, there are the ProcessStarters `MonitorHotfolderExample` and `MonitorHttpLocationExample`, which generate archival packages from files and folders beneath a given hotfolder or URL. A more special one would be a `server` (Server and `ClientLoaderServerThread`), that makes one machine listening to retrieve archival packages from another machine for further processing. It is also possible to run multiple ProcessStarters at the same time by setting more than one value in the configuration parameter `defaultProcessStarter`.

### Starting a WorkflowTool instance

The batch files, which are included within this release, are used to execute an instance of the WorkflowTool after a correct installation. To do this, the batchfiles simply have to be customized with the respective local configurations. Additionally, optional module packages, developed by other institutions can be included and used through the batchfiles (see section 2 on page 7).

`workflowtool -h` explains the command line options:

```
usage: WorkflowTool
    -c, --config-file        The config file to use.
    -s, --show-properties    Prints the system properties and continues.
    -p, --process-starter    The process starter module which chooses items for
                               processing.
    -h, --help               Prints this dialog.
```

Optional to the execution of the batch file, a WorkflowTool instance can also be started through command `java -jar kolibri.jar [OPTIONS]`

Because the access to DIAS is usually realized through an encrypted connection, the parameter `-Djavax.net.ssl.trustStore=KEYSTORE_LOCATION` will be necessary after the installation of the certificate of the DIAS hosting partner with the Java keytool. The path to the keystore file, as well as the path to the `known hosts` file, can be set in the configuration file.

## 3.1 Overview over more ActionModules and ProcessStarters

### 3.1.1 ProcessStarter

`Server` The koLibRI instance is listening for network connections and starts a new thread for each request, that executes the necessary actions. This `ServerThread` has to be set in the configuratiopn file as `serverClassName`, for example `ClientLoaderServerThread` (takes archival packages from other koLibRI instances).

Important configuration values:

- serverClassName
- defaultConnectionPort

`MonitorHotfolderExample` Example ProcessStarter that builds an archival package (Submission Information Package – SIP) for every subdirectory of a given hotfolder directory. It monitors the directory, stated as `hotfolderDir` and processes all existing subfolders first. Then it processes all added subfolders. The names of the subfolders are interpreted as persistent identifiers for the SIPs.

Important configuration values:

- hotfolderDir

- readDirectoriesOnly
- startingOffset
- checkingOffset
- addingOffset

`MonitorHttpLocationExample` Same as MonitorHotfolderExample, but files and directories can be retrieved by an URL. This URL is also monitored and added directories are processed. Directory names are interpreted as persistent identifiers as before.

Important configuration values:

- urlToMonitor
- readDirectoriesOnly
- startingOffset
- checkingOffset
- addingOffset

### 3.1.2 ActionModule

`AdaptHtmlPage` Exchanges links in HTML files. This action module is used by the `MigrationManager` to change filenames to those of migrated and most likely different filesnames, e.g. to keep an HTML page and its linked images useable.

Important configuration values:

- htmlPageFilename
- replaceFulltextOnly

`AddDataToDb` Adds configurable information to the database, if `useDatabase` is set in the configuration file.

Important configuration values:

- storeFileData
- storeFileDataTechMd
- storeDc
- storeIds
- storeCustomData
- customDataEelements

`AlreadyIngestedChecker` Using the XORed checksumme from the koLibRI database the `AlreadyIngestedChecker` checks if a SIP already was ingested.

Important configuration values:

- waitUntilNextTry
- errorIfChecksumExists

`AskDIASifIngested` Tests via DIAS access, if a SIP with a certain external ID already was ingested before.

`CleanPathToContentFiles` Deletes the files and folders in the temporary processing directory.

`CleanUpFiles` Deletes created ZIP files and METS files from the destination directory.

`DiasIngestFeedback` Uses the DIAS ticketing system to request the status of an SIP. All data put in the koLibRI database for this SIP can be removed if an ingest was unsuccessful. Use the config values below to configure what table entries to delete.

Important configuration values:

- removeFileOnError
- removeTechmdOnError
- removeDublincoreOnError
- removeIdentifierOnError
- removeVariousOnError
- removeDiasOnError

`Executor` Executor provides the functionality to invoke system based command line tools. The static arguments have to be set in the configuration file.

Important configuration values:

- command

`FileCopyBase` Files and folders of the object are copied to a temporary processing directory, to work with these copies, without altering the original files. The ActionModul can be extended by own modules, i.e. to change the names of the copied files or ignore certain files for the archival package.

`FileCopyHttpBase` Copies files and folders over HTTP. This allows archiving files from a webserver. For more, see `FileCopyBase`.

`MetadataExtractorDmd` Includes descriptive Metadata into the archival package. This module has to be customized for individual needs. The given example only shows how descriptive metadata is embedded into a SIP. The source of this metadata is dependant of the structures of the individual institutions. Inherits from `MetadataExtractorBase`.

`MetadataExtractorDigiprovmd` Includes provenance metadata into the archival package. This module also has to be customized for individual needs. Inherits from `MetadataExtractorBase`.

`MetadataGenerator` Generates technical metadata for all content files of the archival package with the use of the JHOVE toolkit from the Harvard University Library (please see section 10.1 on page 51 for further information).

Important configuration values:

- acceptedUnknownFileFormat
- showGifGlobalColorTable
- showPdfPages
- showPdfImages
- showPdfOutlines
- showPdfFonts
- showMixColorMap
- showTiffPhotoshopProperties
- showWaveAESAudioMetadata
- showHtmlLinks
- showHtmlImages
- showIso9660FileStructure
- jhoveIsVerbose
- errorIfSigmatchDiffers

`MetsBuilder` Creates a METS file for the archival package out of the gathered informations. Responsible for the creation is the class, stated in `mdClassName`. For kopal, the implementation `Uof.java` of the interface `MetadataFormat` creates a METS file according to the UOF of the kopal project.

`MigrateFiles` Used by the MigrationMagager to convert files in another – likewise newer – file format.

Important configuration values:

- migrationToolExecutionCommand
- sourceFileName
- destFileName

`PrepareMigration` This module extends the UOF with the provenance metadata needed for a migration.

Important configuration values:

- migrateMetadataRecordCreator
- provmdComments
- provmdPermission
- provmdCreator
- provmdPurpose
- provmdResult
- provmdSteps

`SubmitDummySipToArchive` Example module for demonstration purposes.

`SubmitSipToClientLoader` Submits a SIP to a client loader. This module can be used by more than one asset builder at the same time, that send their archival packages to a central WorkflowTool instance. Please see "Informations about the example configuration" for further information.

Important configuration values:

- clientLoaderServer
- clientLoaderPort
- submitPolicyName
- fileSubmitNotificationTime

`SubmitSipToDias` Transfers a SIP to the DIAS, used by kopal, and is the implementation of the SIP specification [10] for ingests into kopal-DIAS, provided by IBM. Some of the following configuration values are set in the common section of the main config file.

Important configuration values:

- cmUser
- cmPwd
- ingestPort
- ingestServer
- knownHostsFile
- preloadArea

`TiffImageMetadataProcessor` Validates all TIFF image files within the archival package with the use of JHOVE first, see section 10.4, page 70.

Important configuration values:

- correctInvalidTiffHeaders
- verbose

- separateLogging
- separateLogfileDir
- jhoveConfigFile
- nonVerboseReportingTime

`Unzip` Un-zips a ZIP file into the given directory.

Important configuration values:

- tempDir
- sourceFile
- deleteZipOnExit

`Utf8Controller` With the help of this module, a created METS file, that is always UTF-8 encoded, can be checked for invalid UTF-8 characters and correct them. The source of the used Utf8FilterInputStream is the utf8conditioner by Simeon Warner (`simeon@cs.cornell.edu`), which was adapted to JAVA for the use in koLibRI.

Important configuration values:

- filename
- storeOriginalFile
- originalPostfix

`XorFileChecksums` Generates an XOR checksum from all existing file checksums and puts it into the database. This XOR checksum can be used to check if an identical SIP has already been ingested into the archive.

Important configuration values:

- errorIfAlreadyIngested

`Zip` Compresses the content files together with the METS file (mets.xml) into a compact package.

Important configuration values:

- compressionLevel

Additional – partly very specialised – ActionModules and ProcessStarters are currently available by request from the kopal project partner The German National Library (DNB) and the Goettingen State and University Library (SUB).

## 3.2 Get started! – Information about the example configuration

There are three ways to use the example configuration files. The first is to configure a single machine, so that it creates and ingests archival packages into an archival system itself – a combination of SIP creation and SIP transfer with one instance of the `WorkflowTool` class.

There is also the possibility to configure multiple machines as so called `AssetBuilder` and let them create archival packages resulting of different workflows and submit them to another central machine. This central machine, the so called `ClientLoader`, receives all these archival packages and ingests them into an archival system – a separation of the SIP creation and the SIP transfer with one instance of the WorkflowTool as ClientLoader, and multiple instances as AssetBuilders.

The third way is to use a database for bookkeeping over the ingested and created archival packages. This has currently only been tested further with a single instance of the WorkflowTool in standalone mode (see first point).

ATTENTION: The use of the database has not yet been tested for the parallel use of multiple WorkflowTools as standalone AssetBuilders. Usage for multiple independent instances of the Workflowtool in standalone mode is theoretically possible, but must be used at one own's risk. Usage of the database through a ClientLoader is not yet possible.

### 3.2.1 Standalone mode – Creation and ingest of archival packages with a single instance of the WorkflowTool

The file `example_config_standalone.xml` is used for a single instance of the class WorkflowTool. To execute this example, all that must be done, is to simply fill all tags enclosed by stars (***) with the right values. These are four path values for the work and temporary directories, a URL or path value depending on the ProcessStarter, and maybe a path for more logfiles. The `description` tags in the configuration file give a description of the according values.

There is the choice out of two ProcessStarters at the moment. `MonitorHotfolderExample` takes all subfolders of the directory stated in `hotfolderDir` and further processes all subfolders added later as an archival package to ingest (SIP), by adding the subfolders to the `ProcessQueue` and processing them. `MonitorHttpLocationExample` does the same, but checks files and subfolders of an URL stated in `urlToMonitor`. Of course the stated directories must exist and contain files and folders, of which JHOVE can extract technical

metadata, to get meaningful and usable results. Any kind of file structure and files can be used to test this module. See section 10.1 for further information.

The policy form the policy configuration file `policies.xml` that is used here, is `example_standalone` and executes the following ActionModules for each step. More informations for each module can be obtained through the documentation of the Java classes or the `description` tags in the configuration file (section 3.1, page 13).

- XorFileChecksums
- TiffImageMetadataProcessor
- MetadataExtractorDmd
- MetadataGenerator
- MetadataExtractorDigiprovmd
- MetsBuilder
- Utf8Controller
- Zip
- AlreadyIngestedChecker
- AddDataToDb
- SubmitDummySipToArchive
- CleanPathToContentFiles
- CleanUpFiles

The command to run the standalone example from a command line is as follows:

```
java -jar kolibri.jar -c config/example_config_standalone.xml
```

or better (if `workflowtool.bat` or the `workflowtool` script has been configured), to set some more, maybe important parameters:

```
workflowtool -c config/example_config_standalone.xml
```

### 3.2.2 ClientLoader/AssetBuilder constellation – Separation of the creation and ingest of archival packages with two or more instances of the WorkflowTool

The file `example_config_clientloader.xml` is used with a single instance of the class WorkflowTool, the file `example_config_assetBuilder.xml` can be used for one or more instances of the WorkflowTool each as AssetBuilder. For the communication between the instances,

some more configuration values are needed in the configuration files. The division of work is described below.

The AssetBuilders can be started on different machines. This way, each machine can process an individual workflow for the creation of archival packages. For individual workflows, also individual policies and configuration files have to be created. The AssetBuilders in this example process the same steps as in the last example. Again, both ProcessStarters stated above, can be used. The difference is, that the created SIPs are not directly transferred to an archival system by each AssetBuilder himself, but are submitted to a central ClientLoader first. To do this, the AssetBuilders use the ActionModule `SubmitSipToClientLoader`. The policy is `example_assetBuilder`.

A WorkflowTool instance with the ProcessStarter `Server` is started as ClientLoader. It is then waiting for request on a port, defined by the field defaultConnectionPort in the configuration file. When the server receives a request from an AssetBuilder, it starts a ClientLoaderServerThread, configured by `serverClassName`. These Threads are listening for the transfer of a SIP from the requesting AssetBuilder. Multiple SIPs from more than one AssetBuilder can be received at the same time.

The policy here is `example_clientloader` and contains two ActionModules:

`SubmitDummySipToArchive` Is the same ActionModule as in the above example. The separation of AssetBuilders and ClientLoaders is suggestive if multiple AssetBuilders shall be used, e.g. to process different workflows at the same time and create SIPs from different sources. The ClientLoader gathers all archival packages from all AssetBuilders centrally, and can also apply additional actions on them, e.g. burning CD-ROMs, revalidating the METS files or inform other institutions about newly ingested SIPs.

`CleanUpFiles` After successfull ingest into the archive system, the archival packages, delivered by the AssetBuilders, are deleted. Commented out in the example for better understanding.

The command to start the AssetBuilder/ClientLoader example on a command line is as follows:

1. Starting the ClientLoader:
   ```
   java -jar kolibri.jar -c config/example_config_clientloader.xml
   ```

2. Starting an AsserBuilder:
   `java -jar kolibri.jar -c config/example_config_assetbuilder.xml`

3. Starting more AssetBuilders:
   `java -jar kolibri.jar -c config/example_config_assetbuilder.xml`

or better (if `workflowtool.bat` or the `workflowtool` script has been configured), to set some more, maybe important parameters:

1. Starting the ClientLoader:
   `workflowtool -c config/example_config_clientloader.xml`

2. Starting an AsserBuilder:
   `workflowtool -c config/example_config_assetbuilder.xml`

3. Starting more AsserBuilder (optionally with different configuration files):
   `workflowtool -c config/example_config_assetbuilder.xml`

### 3.2.3 Standalone mode using the database – Creation and ingest of archival packages with a single instance of the WorkflowTool using the database

First a database has to be created, according to the database documentation. If this was successful, the following steps have to be done:

1. The WorkflowTool has to be configured as described in point 3.2.1.

2. The tag `useDatabase` of the configuration file has to be set to `TRUE`.

3. All database related values have to be filled with the right contents. These values are marked by ### in the configuration file.

4. The WorkflowTool can now be started as described in point 3.2.1. The database is now updated with useful information during the ingest process. See the database documentation for further information about the use of the database.

If the database shall not be used, all that has to be done, is to set the value of `useDatabase` to `FALSE`. The modules that use the database can remain in the policy file, because all database modules check this parameter.

# 4 Using koLibRI for retrieval

The functionality of retrieving archive objects is usable via the Web Service (see chapter 9 on Web Service). On the other hand, one can use the command line tool `DiasAccess`, and the according Java methods of the same class within own software respectively. For better understanding of the DiasAccess command line options, knowledge of the DIP interface specification [11] is important. An overview over these options can be found in section 10 on page 51.

# 5 The Migration Manager

## 5.1 Description

The MigrationManager is a koLibRI component which manages and executes migrations. Objects which have to be migrated and those which should be result of the migration can be described. Depending on those objects and your own requirements individual migration workflows can be configured and different migration tools can be used.

The MigrationManager is a prototype and the current functionality is between the simple and complex scenarios described below. It is not sufficiently tested to be used for the productive mass use and not all possibilities of DIAS, koLibRI and the UOF are implemented. But for exemplary test operations and for conception of migration processes the MigrationManager can be regarded as usable.

Migration is in itself a demanding process in the operations of an archive which can not be isolated from the other processes and plannings of the archive operations and the available infrastructure. A very simple scenario which can be implemented without problems with the current version would be:

- The source file format TIFF and destination file format JPEG2000 are entered on the command line,
- the matching objects are looked up on a set of local available METS files, and
- a simple migration workflow consisting of downloading the affected archival packages, migration of files with a command line progamm and the new building and ingesting of the archival packages is executed.

A complex scenario which can not be implemented because of missing supra institutional infrastructure and know-how, but for which the basic architecture and interfaces for further development are present:

- A koLibRI instance receives a message from a technology watch service or format registry (like GDFR [26] oder PRONOM [27]) that an image format produced by a specific programme and with specific technical properties has to be migrated into another format,
- the objects and files with the matching properties are looked up in the DIAS data management,
- because the affected files and objects are templates used by other objects, some objects have to be migrated and adapted together,

- in some files the affected image formats are embedded,
- an external service registry is queried for an appropriate migration service (e.g. a grid service for the conversion of big data sets), and
- the depending files and objects are adapted.

## 5.2 Design

The MigrationManager itself is a `ProcessStarter` which consists of several functional subentities:

`MigrationEventListener` Fetches or receives messages about necessary migrations. This happens currently with the command line where the characterisation of source and destination objects is specified.

`ObjectCharacter` A characterisation of objects like e.g. source and destination file format but also more complex properties like producing software or encoding. Because there is no standard for the technical description of objects the kopal UOF is used as base and single properties are defined by XPath expressions. An object characterisation through CQL [25] for the DIAS data management is also possible.

`ObjectCharacterTranslator` Translates object characterisations into queries of a data management system. For XPath object characterisations this is possible through queries on METS files in the local file system, on the koLibRI database or the DIAS datamanagement. Only the DIAS datamanagement is capable of CQL object characterisations at the moment. It has to be kept in mind that not every data management does support all kinds of queries. Complete in the sense of the kopal UOF is the querying of local METS files.

`ObjectDependencyAnalyser` Analyses dependencies between objects and combines them. At the moment the `mptr` (METS pointer) of METS files are analysed and all objects, connected by METS pointers, are collected. The further processing of dependent object is not used at the moment.

`MigrationPolicyBuilder` Configures migration processes with the help of templates. Depending on the template and the source and destination object characterisation the appropriate migration service is chosen.

Besides the configuration possibilities in the common configuration file, two more configuration files are important for the MigrationManager:

`migrationPolicyTemplates.xml` Within this file (or in a file named according to the common configuration) the policy templates according to which the migrations are performed are defined .

`migrationServiceRegistry.xml` The available migration services are listed here as well as defined how they can be used.

The most important action modules for the MigrationManager are:

`PrepareMigration` Adapts the UOF for an migration. This module has to be called very early in the migration process or respectively before the file conversion.

`MigrateFiles` Executes external programmes for the migration and adapts the metadata into the UOF accordingly. Migrations are only possible through execution of command line programmes currently.

`Executor` Executes external programmes.

`AdaptHtmlPage` Adapts links and file references within a HTML page if the name of a file has changend through migration.

## 5.3 Usage

To perform a migration, several settings have to be configured and the migration process has to be triggered through a migration event.

### 5.3.1 Configuration

1. Which data mangement has to or can be used? The corresponding `ObjCharacterTranslator` class has to be specified in the configuration file as parameter `ObjChrTranslatorClassName` for the class `processstarter.migrationmanager.MigrationEventScheduler`. If direct access to DIAS is available `DiasCQLObjCharacterTranslator` can be used for a CQL characterisation or `DiasObjCharacterTranslator` can be used as XPath characterisation. Alternatively `FileObjCharacterTranslator` is usable for locally available METS files or `DbObjCharacterTranslator` for the koLibRI database.

2. For `FileObjCharacterTranslator` the `MetsFilesDir` in which the directories or sub directories with METS files are stored has to be specified.

3. For the class `processstarter.migrationmanager.MigrationPolicyBuilder` the `migrationServicesFileName` and `migrationPolicyTemplatesFileName` have to be given as paths to the configuration files which define available migration services and workflows.

4. The usable migration services have to be defined in the configuration file named `migrationServicesFileName`. Descriptions and examples of the configuration can be found in the file `config/example_migrationServiceRegistry.xml`. Two aspects are essential: It has to be described through which action module and with which parameters a service can be called. It also has to be defined which properties the service has got, so that it can be chosen for a specific migration. This is possible by name, by classifications or by source and destination object characterisations using XPath expressions (e.g. source and destination file format).

5. The templates for migration workflows have to be defined in the configuration file given as `migrationPolicyTemplatesFileName`. Descriptions and examples of the configuration can be found in the file `config/example_migrationPolicyTemplates.xml`. It is also essential here, that application criteria of a migration template are defined. The criteria are also expressed as XPath object characterisations. After that follows the definition of the policy parts (`policyFragments`) to use. Fixed policies of the `policies.xml` (e.g. for the invariant parts of the ingest process) or criteria for choosing a migration service listed in the service registry are listed in order of execution.

### 5.3.2 Executing the Migration

If all necessary configurations are done, the migration manager can be started by invoking koLibRI with the class `MigrationEventScheduler`. Two parameters are expected if only a source and destination format shall be used. This is done by providing the according DIAS format IDs. If four parameters are provided, the first pair is interpreted as XPATH expression and its expected return value for the source object characterisation. The second pair respectively for the destination object characterisation. An object characterisation can be done in longer form by

```
/mets/amdSec/techMD/mdWrap/xmldata/format/text()
```

(namespaces are left out in this example – due to better readability – but however have to be used) or in a more practicable form by

```
//*[name()='lmerFile:format']
```

As a result, GIF (for example) can be defined as

`urn:diasid:fty:kopal:020050705000000000005`

All objects containing GIF files are selected by this.

## 5.4 Class Descriptions an Extensions

- `MigrationEventScheduler`
  Implements the interface `ProcessStarter`, initialises all further components with help of the configuration and manages the main process.

- `MigrationEventListener`
  Interface for classes that receive events, that require a test for the need of a migration.

- `CommandLineMigrationEventListener`
  Implements the interface `MigrationEventListener`. Interprets commandline arguments as events.

- `ObjCharacter`
  Object characterisation through XPATH expressions. Enables the usage of the `FileObjCharacterTranslator`, `DbObjCharacterTranslator` and the `DiasObjCharacterTranslator`.

- `CQLObjCharacter`
  Object characterisation through CQL expressions for the DIAS data management. Requeires the usage of the `DiasCQLObjCharacterTranslator`.

- `ObjCharacterTranslator`
  Interface for the translation between object characterisations into queries of according data managements.

- `FileObjCharacterTranslator`
  Queries as quasi-data-management single or multiple METS files. This also qualifies it for the mapping of XPATH queries into queries of other data management systems. This can be done by querying METS files with the `FileObjCharacterTranslator` that contain the fieldnames of the according data management instead of the regular values of an archive object.

- `DbObjCharacterTranslator`
  Queries the koLibRI database as data management, if the objects characterised through

XPATH exspressions are present. For the mapping it uses the class `FileObjCharacterTranslator` and the file `MapUofToDB.xml`.

- `DiasObjCharacterTranslator`
  Queries the DIAS data management, if the objects characterised through XPATH exspressions are present. For the mapping it uses the class `FileObjCharacterTranslator` and the file `MapUofToDIAS.xml`.

- `DiasCQLObjCharacterTranslator`
  Queries the DIAS data management for objects characterised through CQLObjCharacter.

- `ObjectDependencyAnalysator`
  Analyses dependencies between objects and bundles them. Currently, only objects linked through METS pointers are analysed.

- `MigrationPolicyBuilder`
  Builds and configures migration policies with the help of the classes `MigrationPolicyTemplateStore` and `MigrationServiceRegistry` for given objects and object characterisations.

- `MigrationPolicyTemplateStore`
  Class that delivers the migration templates for queries on basis of the according configuration file.

- `MigrationServiceRegistry`
  Class that delivers descriptions and parameters for chosen migration services on basis of the according configuration file.

# 6 The koLibRI database

koLibRI uses the functionality of Hibernate [12] for all database access to be independent of one certain database system. To use another system instead of MySQL, just change the database system and also the values `hibernateConnectionDriver_class` and `hibernateDialect` stated in the main config file. For further information please consult the Hibernate documentation.

### Current limitations

- Parallel database access from different machines has not been sufficiently tested yet.
- The koLibRI database uses foreign keys, transactions and auto-incremental functionalities, which limits the choice of the table engine to `InnoDB` when using MySQL.

## 6.1 Installation

### Required software/Jars

The class path has to include (all needed packages are located in the `/lib` directory):

- `db-beans.jar`
- `hibernate3.jar`, included are packages required by Hibernate
- a jar of the database driver for MySql, e.g. mysql-connector-java-5.0.6-bin.jar

### Installation of a database

The koLibRI database was tested with a mySQL database in version 5.0 and the InnoDB table engine, contained in this version. With *mySQL Administrator* and *mySQL Query Browser* there are easy to use and free managing tools available, that can do many configuration and query tasks. Please read the according installation guides for all databases.

### Creation of a database schema

The database schema for mySQL can be created with help of the SQL script `createDB.sql` (under `config`). `kolibriDbTest` is used as default database name, this can customized by search and replace functionalities in the script. For executing the script, either the Query Browser can be used, or you can connect through command line by

```
mysql -u [USER] -h [HOST] -p
```

and execute

```
source [PATH]/createDB.sql
```

To use the database, it is important to fill the tables owner and server with at least one entry each, which then have to be set in the configuration file as `defaultOwner` and `defaultServer`. A test owner and server already are put in if you use the `createDB` script. For other databases, a specific version of the createDB script has to be created.

### Configuring the database

A database user and password for koLibRI must be created, that is allowed to access the database with full rights (except DROP) from all machines running koLibRI. This can be done under `User Administration` in the mySQL Administrator or using the command line. Firewall settings do also have to be considered for remote access.

### Configuring koLibRI for the database

For the use of the database, koLibRI has to be configured at various points. In addition to the setting if the database is even used (and not only one ore more log files), there also have to be set class specific values for `util.db.HibernateUtil` in the configuration file (see config file). Besides that, the moment and type of the stored data has to be declared for the according policies. This is done by addition of the ActionModule `actionmodule.sub.AddDataToDb`. The following options are available:

- `storeFileData`
  fills tables *file* and *fileformat* for each file in the metadata class.
- `storeFileDataTechMd`
  fills table *techmd* for each file in the metadata class.
- `storeDc`
  stores all Dublin Core metadata of the metadata class in table *dublincore*.
- `storeIds`
  stores all entries in `ProcessData.customIds` in table *identifier*.
- `storeCustomData`
  stores some entries in `ProcessData.customData` in table *various*.
- `customDataEelements`
  states the elements in `ProcessData.customData` that shall be stored.

The ActionModules `actionmodule.AlreadyIngestedChecker` (before ingest) and `actionmodule.DiasIngestFeedback` (after ingest) can be used to add entries into the *dias* table. These do some checking and only ingest the packages, or respectively add data to the database, if these checks were successful. If you are using the three modules as in the example policies, you can first add the data to the database, and then ingest the SIP. If the ingest was *not* a success, you can configure the data to be deleted afterwards from the database. `DiasIngest` is not being used in the example files. It can only be used to document the functionality, as direct access to DIAS is required for its functionality.

All parameters needed for the proper configuration can be looked up in the config file `config.xml`.

## 6.2 Design of the database schema

The *input* table contains files, that koLibRI received at "'first contact"' with the source material: Who is the owner of the object (reference to *owner*), where does the request or files come from (referende to *server*) and when (column *startdate*, given in milliseconds since 1970, see `java.lang.System.currentTimeMillis()`). If there are files present in the ProcessData object even before the execution of the policy (that is after the ProcessStarter), an XORed SHA-1 checksum of all files is stored. In addition, the object can have multiple files, Dublin Core metadata, identifier or extra information (references to *file*, *dublincore*, *identifier* or *various* on the *input* entry).

Warnings and errors may appear while processing the policy. Such events are listed in table *event* and refer to the *input* entry, that the problem is related to. There are references to the *modulestatus* and the *description* of the error. In addition, the time of the event (*eventdate*) and the module name (*module*) are being logged. There is also the possibility for a reference to the file that was responsible for the event.

Finally, the archival package gets successfully ingested into DIAS, and an entry into *dias* is made. Next to the external and internal DIAS IDs and the time if the ingest, the internal ID of the parent object is also logged for a migration. If the files were changed since their entry to the *input* table, the altered XOR SHA-1 checksum of all files is stored here. The *deleted*-flag is used to also be able to log deletions.

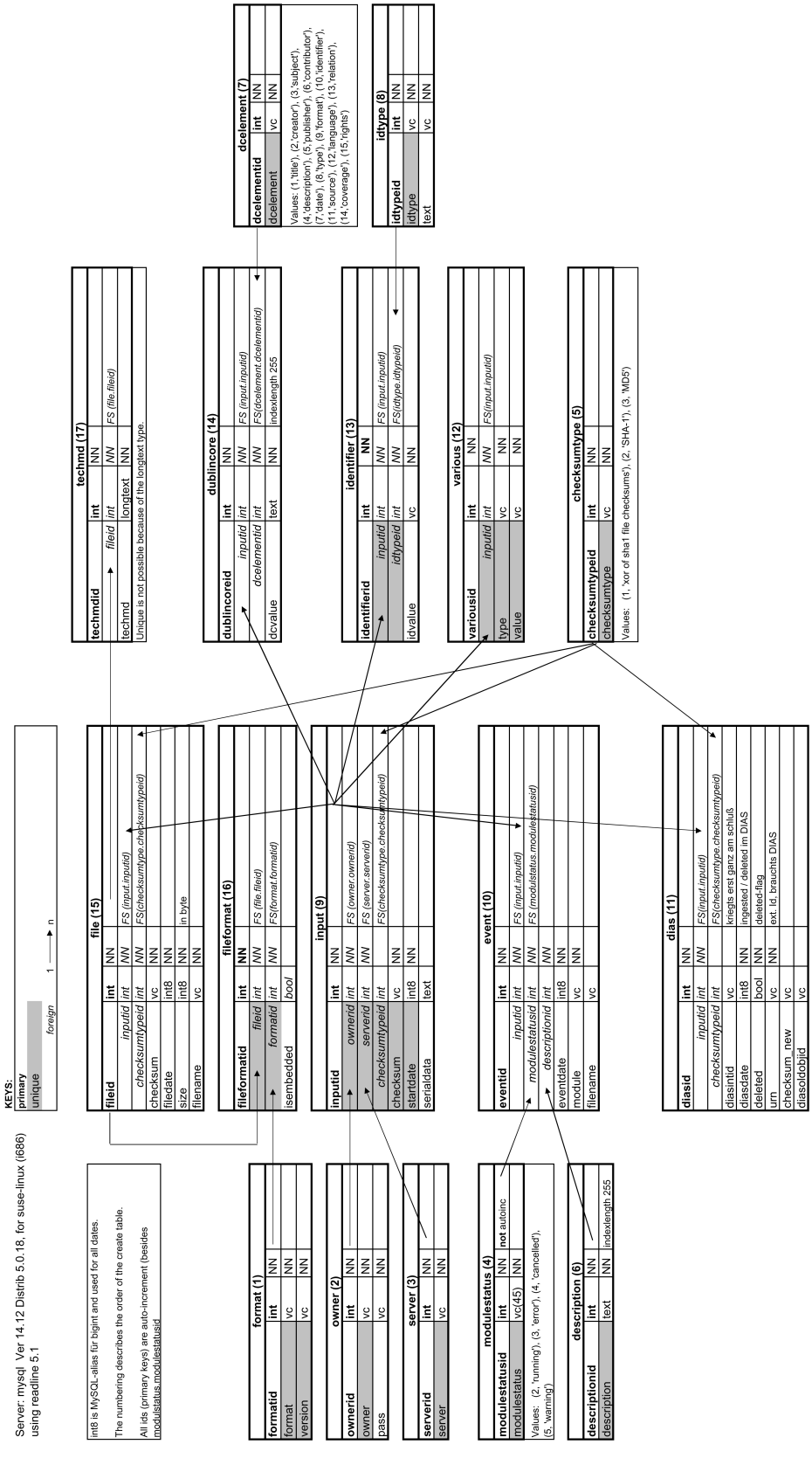


Figure 2: The koLibRI database schema

# 7 Customized koLibRI extensions

For further extensions and customizations, the interfaces `ActionModule` and `ProcessStarter` become more interesting in particular. A certain familiarity with the structure of koLibRI and the classes of the workflow framework `de.langzeitarchivierung.kopal` is, however, necessary. At this point, a functional overview shall be provided. Please consult the javadoc API documentation for more detailed information.

## 7.1 The Structure of koLibRI

The package structure of koLibRI is lying within `de.langzeitarchivierung.kopal`. This package contains the central classes of the workflow framework. Underneath lie the following packages. It is possible to create subpackages for institution specific extentions.

- `actionmodule`
  Contains the `ActionModule` interface and its implementing classes.
- `administration`
  Contains all classes implementing the search and administration interface of DIAS, see sectiob 8, page 40.
- `formats`
  Contains the interface `MetadataFormat` in its implementing class Uof for the *Universal Object Format* of the kopal project.
- `ingest`
  Classes for the ingest into DIAS.
- `jhove`
  Additional classes for the *JSTOR/Harvard Object Validation Environment*, implemented by the DNB and SUB.
- `processstarter`
  Contains the ProcessStarter interface and its implementing classes.
- `retrieval`
  Classes for the access to assets within DIAS.
- `ui`
  Classes for user interfaces and event handling.

- `util`
  Miscellaneous helper classes, e.g. methods for file handling, HTML- and HTTP-access, the TiffIMP, etc.

- `ws`
  The koLibRI Web Service, please see section 9.

The workflow framework `de.langzeitarchivierung.kopal` is consisting of eight classes at the moment:

- `Policy`
  Manages a workflow and builds it from ist XML representation.

- `PolicyIterator`
  Iterates over the working steps of a workflow tree.

- `ProcessData`
  Contains the central information of an asset: Name of the process (e.g. the assets URN), policy, metadata, file list. The run method contains the logic for processing the policy.

- `ProcessQueue`
  A queue of all `ProcessData` objects to be processed.

- `ProcessThreadPool`
  A ThreadPool, that processes ProcessStarter and ProcessData objects.

- `Status`
  Contains status value and description for the actual state of the ActionModule. The processing of the workflow is controlled by the status values.

- `Step`
  Single working step within the workflow of an asset. Contains – among others – functions to load and start action modules and to set their status values.

- `WorkflowTool`
  Provides the command-line interface, starts the process starters and is working off the processes. Working steps of a process are only processed if its value is `Status.TODO` and its predecessor ended with the status `Status.DONE`.

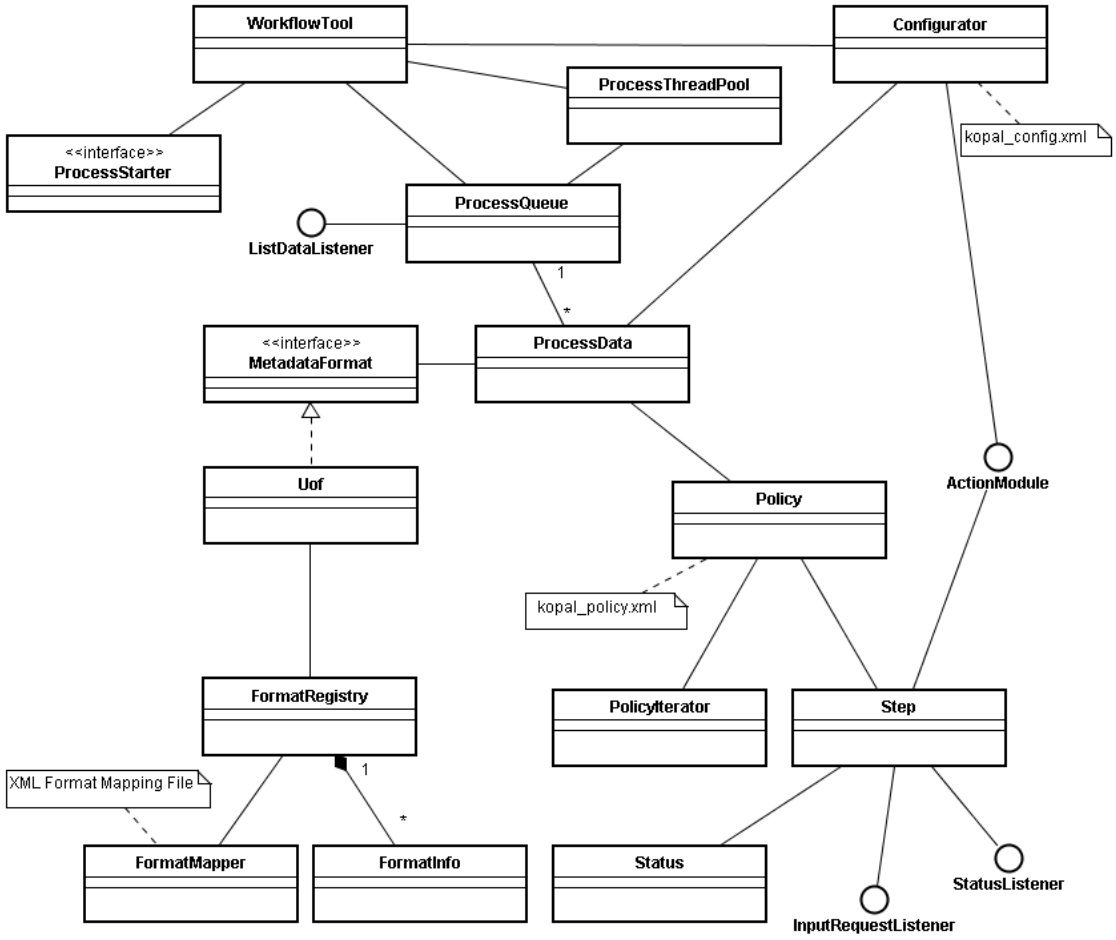


Figure 3: Das Klassendiagramm koLibRI

### 7.1.1 ProcessStarter

For each new SIP, the ProcessStarter has to process the following actions:

1. Create a new ProcessData object,
2. initialize start values of the metadata if necessary (if ActionModules need those for further processing), and
3. add the ProcessData object to the process queue (`ProcessQueue.addElement()`).

As long as at least one ProcessStarter is running, or the ProcessQueue still contains elements respectively, koLibRI does not end, and instead waits for new processes or the end of the actual processed one.

### 7.1.2 ActionModules

An ActionModul has to

1. set its status value to `Status.RUNNING` upon invocation, and
2. set ist status value to `Status.DONE`, when it was finished successfully.

An ActionModule should ...

- ... set its status value to `Status.ERROR` if an error occurs. The error message will be logged seperately and modules with this status are not processed again. It is *not* equal with the throwing of a Java-Exception, that means the module is not interrupted in its processing and could, for example, retry the erroneous action and reset its status at a success. A usual procedure would be the setting of `Status.ERROR` on a timeout, directly followed by the `Status.TODO`. The module would then return temporarily to the workflow framework. This way, the error will be logged and the processing will be retried later.
- ... use the staus value `Status.WARNING`, if an event appears, that should be logged as warning (also in the database), but not lead to the abortion of the modules.

An ActionModule can ...

- ... especially work with and update the metadata and files of the ProcessData object.
- ... store `customData` in the HashMap of the ProcessData object for other working steps, which has no place in the structure behind UOF.
- ... access the configuration data.

## 7.2 Configuring classes

The configuration of classes is done through the class `Configurator`. It gets initialized with the help of a XML file by the method `setConfigDocument()` and does then set the required configuration values of classes and objects, especially ActionModules and ProcessStarters, over various overridden configure methods. This is possible, because the classes and objects to configure are providing setter methods according to the JavaBeans specification.

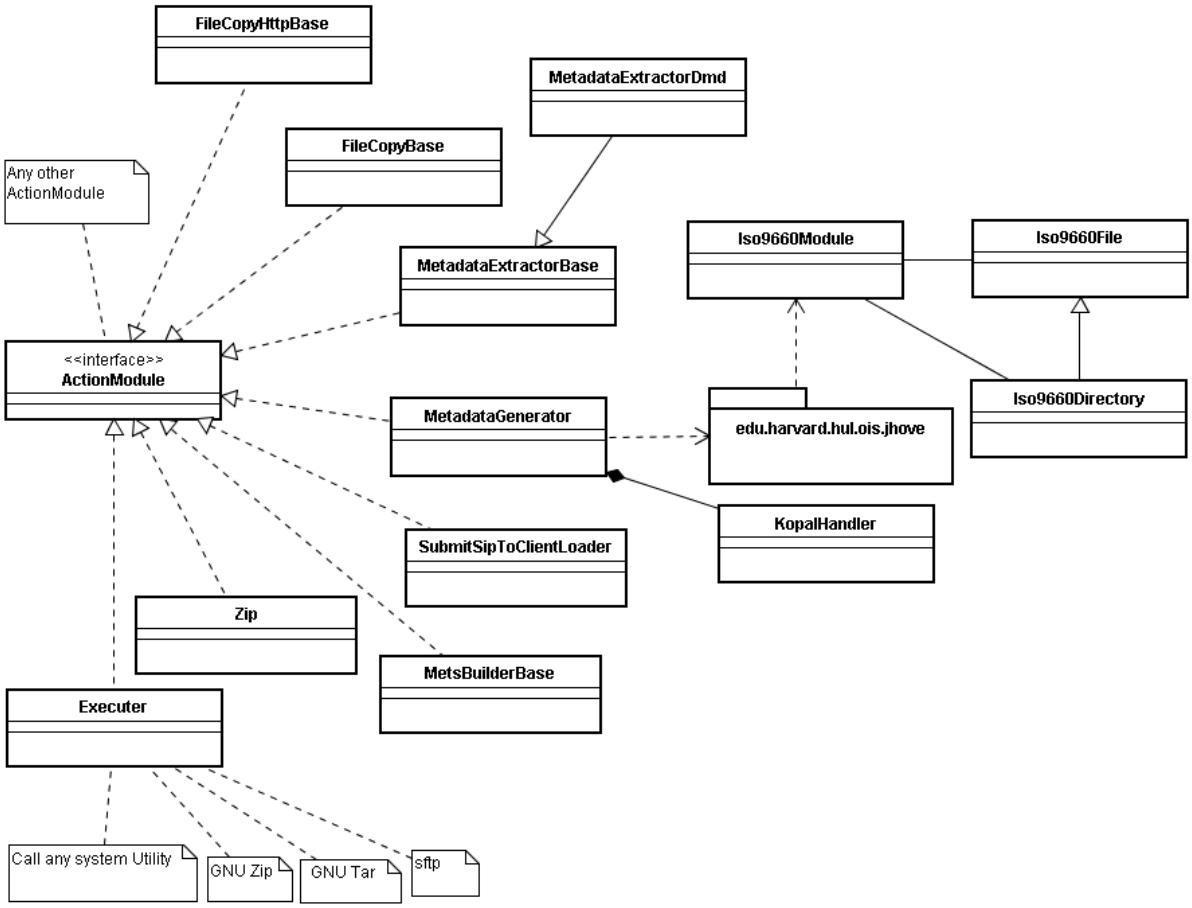


Figure 4: Klassendiagramm ActionModules

If an ActionModule requires the value `useDatabase` from the configuration file for instance, it just provides a `setUseDatabase` method, that is called with the according parameter by the Configurator while execution of the program (see also section 2).

Also possible ist the transfer of configuration values between objects and classes through getter and setter methods. Please see the API documentation for more details.

## 7.3 Metadata formats

The metadata specific functionalities and object structures are mostly concentrated in the `formats` package. The interface `MetadataFormat` declares the basic functionalities that are required for processing. These are not always specific enough to provide full independency of a concrete metadata format like the UOF.

In the current version of koLibRI, this in particular is valid for mapping the metadata for the database by the class `MappToHib`, as by the classes `PrepareMigration` and `MigrateFiles` to handle migration scenarios.

For the implementation of the interface `MetadataFormat`, a Java-XML-Binding through XML-Beans [23] was used. The XML schema, according to the UOF, was transferred to Java classes by the XML-Beans scomp compiler. This allows the structured and consistent usage of the schema in Java. In particular, it provides functionalities for writing, reading and validating of according XML files. To integrate individual schemas or metadata formats, according Java classes can be analogous generated through the XML-Beans scomp compiler. The main work for customization is then primary the implementation of the interface MetadataFormat and additional helper methods, customization of some ActionModules and of the class `MappToHib`, if the database is used.

# 8 Implementation of the DIAS administration and search interfaces

In koLibRI 1.0, classes and interfaces have been added to allow and support the usage of the administration and search interfaces, that were implemented by IBM. To accomplish that, there are methods to generate the according HTTP requests for DIAS, as well as methods and data classes which enable the interpretation and further usage of the XML responses of DIAS.

All classes and interfaces that are required to do that are located within the subpackage `administration`. Its central class is `DiasAdmin.java`. All methods to formulate queries for DIAS and parse its XML responses are concentrated within this class.

DIAS-Core seperates between two interfaces: On the one hand, there is the administration interface, which provides a delete function for objects, various administrative functions for XML Schemas and the list of common filetypes, a consistency check for ingested objects and a function to list all modified objects of a certain timeframe. On the other hand, there is the search interface, which supports queries and responses according to the SRU standard.

The `DiasAdmin` class combines the functionalities of both interfaces, in detail:

- Addition of a new file type to the list of common filetypes in DIAS
- Modification of an entry in the list of common filetypes in DIAS
- Deletion of an AIP (Data of an object for an internal DIAS ID, database entry within dias is however not deleted)
- Deletion of an asset (Complete object for an internal DIAS ID. Deletion, however, is not possible, if the object has migrated children)
- Consistency check for an ingested object
- Addition of a new XML schema for the use within the METS metadata
- Request of a certain XML schema
- Request of the list of supported XML schemas
- Request of a list of objects that were modified within a certain timeframe
- Submition of a formulated CQL query to the SRU server of DIAS
- Preparation of an individual SRU searchRetrieve request for the DIAS server
- Preparation of an individual SRU explain request for the DIAS server
- Parsing of the response from the DIAS administration interface and optional transformation of the data into defined data classes

– Parsing of the SRU response from the DIAS search interface and optional transformation of the data into defined data classes

To use the functionality of this class, there simply has to be added a part for it within the global configuration file. Find a short description of each parameter in the following:

```
<class name="administration.DiasAdmin">
    <field>useHttps</field>
          Specifies wether or not the connection shall be assembled through secure HTTPS.
    <field>adminServer</field>
          The adress of the DIAS administration server
    <field>adminPort</field>
          The port of the DIAS administration server
    <field>adminUser</field>
          The username for administrative tasks
    <field>adminPwd</field>
          The administration password
    <field>keyStoreFile</field>
          Keystore file for a secure connection to the DIAS administration server
    <field>sruMaximumRecords</field>
          (Optional) Specifies the maximum number of records to be returned in a single
          response message
    <field>sruResultSetTTL</field>
          (Optional) Specifies the number of seconds that a result set should be
          mantained before it will be deleted by the server.
    <field>sruSortKeys</field>
          (Optional) Sorting keys for the SRU server to apply.
    <field>sruStartRecord</field>
          (Optional) Specifies the first record of the result set that should be
          returned in the response message
    <field>sruVersion</field>
          (Optional) The version of the SRU server. The default value is "1.1"
    <field>sruX_additionalDataElements</field>
          (Optional) As many DIAS Standard or Custom Data Elements may be repeated
          as needed, seperated by commas. These Data Elements will also be part of
          the SRU response for each record.
    <field>sruX_listAffectedContentFiles</field>
          (Optional) Including this parameter indicates that a list of FileIDs of all
          affected content files (files that meet the search criteria) must be
          included in the response record.
</class>
```

`DiasAdmin` can the be configured through the `Configurator`. Depending on the scope of use, the class can be used within an action module or another java class. The development of a comfortable administration tool with a graphical user interface is of course also

imaginable. At the moment, the class specifically is in use to support the prototype of the *migration manager*. It is used within the classes `DiasCQLObjCharacterTranslator.java` and `DiasCQLObjCharacterTranslator.java` which provide the ability to search objects that have to be migrated over certain search criteria. For further information on that, please see the according part of this documentartion for the *Migration Manager*.

The object-character classes `DiasCQLObjCharacterTranslator` and `DiasObjCharacterTranslator` are responsible for the request of object to migrate through the SRU interface of DIAS.

`DiasCQLObjCharacterTranslator` is using a predefined and complete CQL query, which is boxed into an SRU request and then passed on to the DIAS system. This allows very complex requests. For a definition of possible queries, please consider the official specification of the *Administration and Search interfaces of DIAS-Core*.

`DiasObjCharacterTranslator` is using an search-definition-object of type `ObjCharacter` to define the criteria to search. This enables parallel search within multiple datasources, the kopal database and DIAS itself for example. The mapping file `MapUofToDiasData.xml`, found within the `/config` folder of the release, is used for the translation between the data elements within the UOF-METS file and the data elements of the DIAS-internal database. This file can be easily extended any time.

# 9 koLibRI as a Web Service

## 9.1 Introduction

The ability to act as a web service is one of the main enhancements of version 1.0 against the previous version 0.7. As a web service, koLibRI does not have to be called on the command line to work as a batch process. It is started in a servlet container by the web administrator and runs uninterruptedly listening to predefined port for new requests.

Each request is a SOAP [13] message containing besides the user name, password and institution all the parameters needed to process this request. Java class `WebServiceServer` gets the message and calls the appropriate method in one of the middle layer classes `IngestLayer`, `AccessLayer`, `WSUtils`.

All the services return `ResponseElemDocument` containing mainly a status code, human readable text for that code, DIAS internal ID of this digital object (where needed), time to wait (if DIAS busy), link to the requested file (if there is file to download), extra info string (if necessary), error Message (if any occured) and a custom XML field if the answer contains complex data. So the client can analyse the results of the request after sending each message.

## 9.2 Installation

Installation of koLibRI as a web service is very simple. Just unpack the provided ZIP archive in the Axis2 directory `/WEB-INF/services` and start Tomcat.

Tree files are all you need: `kolibri_web_services.aar` contains all the necessary program code and libraries to run the web service. The other ones are the configuration and policy files in XML format. They are the same configuration and policy files described in previous chapters. For a real productive environment to ingest, there must be another file containing cryptographic certificates.

Installation is so simple if you already have a running Axis2 [14]. If not, you have to install Axi2 first. koLibRI is tested with Axis2 version 1.2. Although Axis2 has its own web server and therefore can run without another web server, it is recommended to run it under a *real* server. koLibRI is tested with Tomcat version 5.5.23 [15].

## 9.3 Configuration

Operating system, Java SDK, Tomcat and Axis2 should be configured as explained in their corresponding documentation. All koLibRI configuration is done in its own configuration file as described in previous chapters.

## 9.4 Starting up

Run Tomcat with the help of `startup` script under `$CATALINA_HOME/bin` directory. Test the functionality of Tomcat and Axis2 by calling their test pages. Use `setParameters` method to set the actual parameters if necessary. You can use `TestWebServices` class as a simple example or write your own code which calls `WebServiceClient`.

Now you are ready to start koLibRI web service. Call `initCache` to initialize the cache – please keep in mind that web service will not function without initializing the cache. After initializing the cache allow Ingest and/or Retrieval by calling `allowIngestRequests` and/or `allowRetrievalRequests` respectively. If no error message returned is, web service is ready for productive work.

Please note that the DIAS installation in GWDG checks the IP address to determine if you are allowed to connect DIAS. So don't be suprised that all your connection requests are refused and you get just errors and exceptions. Use koLibRI to generate SIPs in UOF format but do *not* try to upload it to DIAS.

## 9.5 How Ingest works

As seen in WSDL file, ingest service needs a policy describing how to handle that digital document and one to six metadata blocks of type `customXMLType`. The first one is mandatory and contains – among others – the *must-have* information Persistent Identifier and location of the files belonging to this document. More information like label, checksums, etc. would be better especially for checking the transfer errors but are not mandatory at the moment.

`customXMLType` can theoretically transfer any valid XML. This version focuses on LMER simply because the UOF also uses LMER.

The other five optional metadata blocks are for descriptive metadata in different formats. They will be copied unmodified into METS DescMd section. According to DIAS documentation, if descriptive metadata exist the first one must be in Dublin Core [16].

In this first version of web service ingest is not fully implemented. To generate complete SIPs please use the standalone koLibRI.

## 9.6 How Retrieval works

Sequence diagramm 5 shows the mode of operation of retrieval in time. Typically users search relevant information in library database – which is *not* a part of koLibRI – and have a list of documents as search result. If they think a specific document in this result list is the right one, they choose this document.

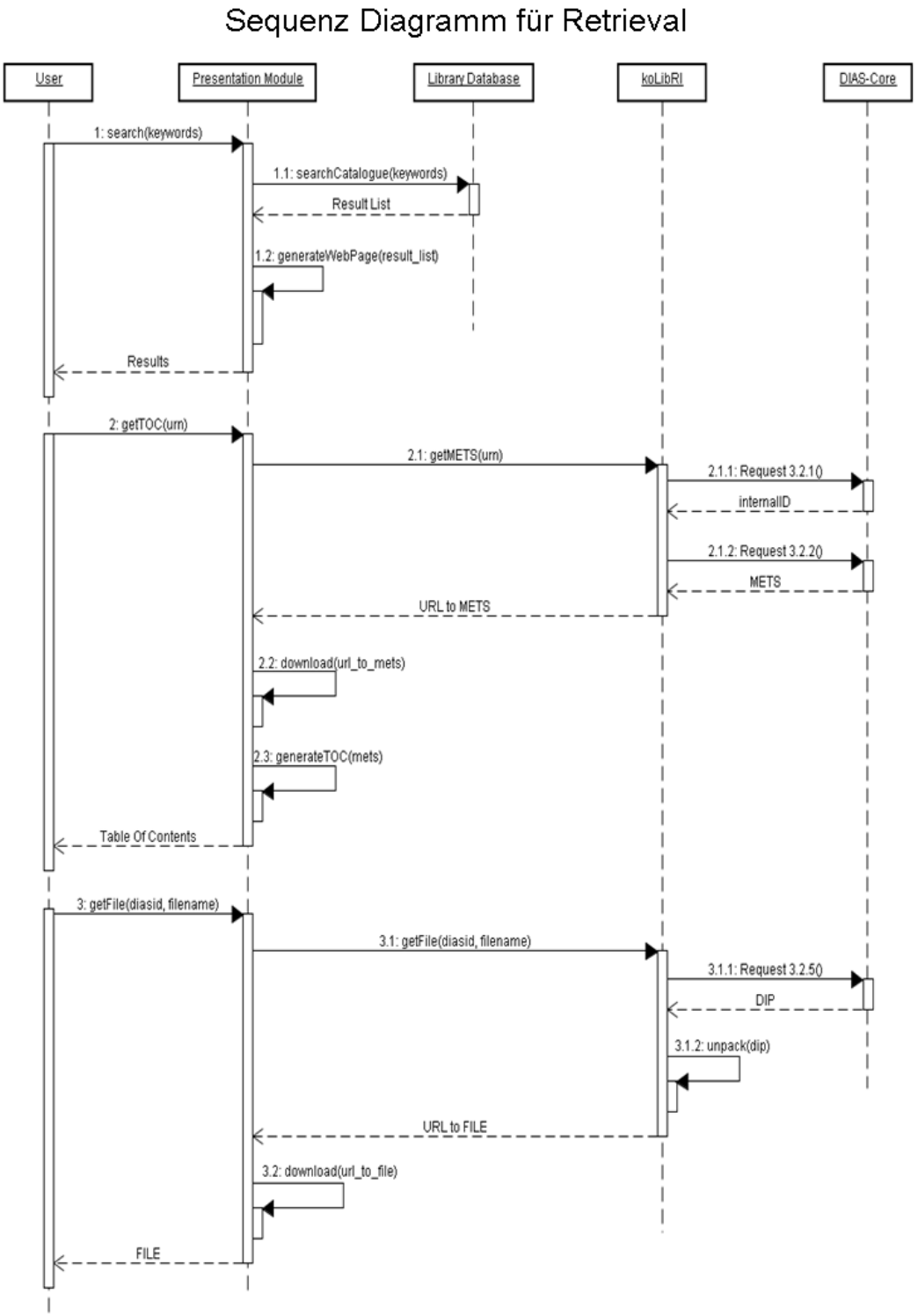


Figure 5: Sequence diagramm of retrieval including external user interface

In case this document is an electronic one, the user interface sends the URN of the document to koLibRI and requests the metadata. Web service requests from DIAS the full metadata for that URN. DIAS responds with a link to the zipped `mets.xml` of that document and its internal DIAS id. Web service generates a unique directory path using this unique internal id and checks there if `mets.zip` is already downloaded. If yes it saves time and resources and sends a link to the saved copy and does not download it again.

The user interface extracts the `mets.xml`, processes it and prepares the user with a table of contents for that digital document. User chooses a specific file from the list. The user interface sends the DIAS internal id and file name to web service. Web service looks in its cache and decides if it must request the file from DIAS or if the file is ready in cache.

## 9.7 Organisation of the cache

Web service uses a directory to save the downloaded files and metadata. This directory is called cache because of its positive side effect that if web service determines that a digital object already has been downloaded a short time ago, this document needs not to be requested and downloaded from DIAS once more. Especially for tape access in DIAS or downloading large files this cache speeds up things treamendously and at the same time reduces the load on DIAS-System.

Actually the cache is not one single directory but a big tree of directories (see figure 6) organised in tree layers and named with digits 0...9. It makes use of the algorithm that the DIAS uses to generate its unique internal id's to determine the location of a file. DIAS genarates for each version of each digital document an unique number in form

`[prefix]:YYYYYMMDDhhmmssSSSnnnn`

where

- `YYYYY` stands for year (02007 for this year)
- `MM` stands for month (01...12)
- `DD` stands for day (01...31)
- `hh` stands for hour (00...23)
- `mm` stands for minutes (00...59)
- `ss` stands for seconds (00...59)
- `SSS` stands for milliseconds (000...999)
- `nnnn` stands for continious number if more than one assets are ingested in the same millisecond.

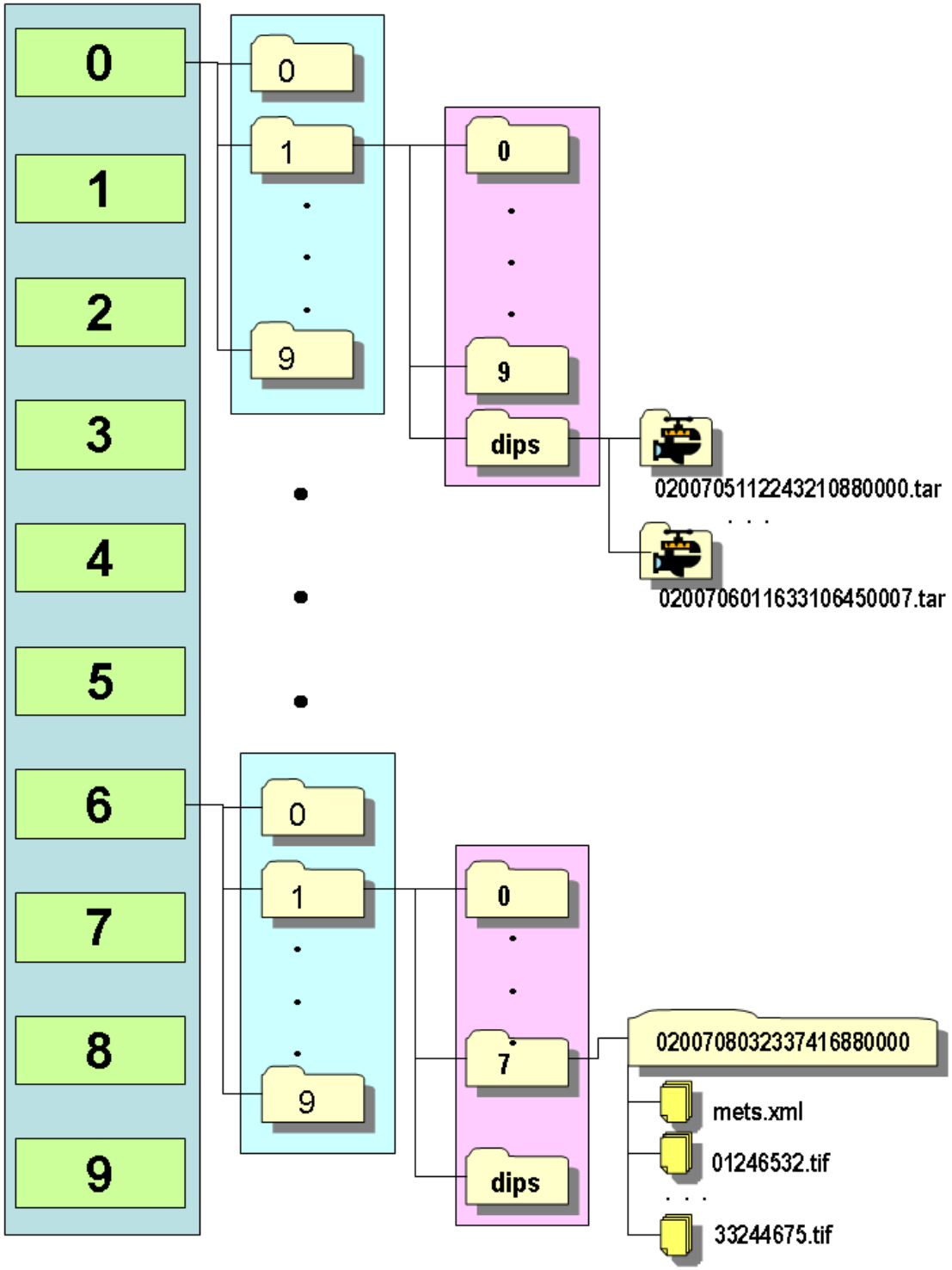


Figure 6: Organisation of the directory structure in cache

Web service takes this id and generates a unique path for that asset so its anytime possible to look in the directory tree to see if that document in that version has been already downloaded. The administrator shall control storage media often to be sure that it is not full. Web service has no automatism to delete the old entries. It offers the `eraseOldCacheEntries` message to be called if necessary.

A more intelligent cache organisation is possible but would be much more complex without a significant win.

## 9.8 Available operations

Web service offers many operations:

- `getFileFormatList`
- `reloadFileFormatList`
- `ingestDocument`
- `ingestSIP`
- `blockNewIngestRequests`
- `allowIngestRequests`
- `getIngestStatistics`
- `resetIngestStatistics`
- `getMETS`
- `getFile`
- `getDIP`
- `getDIPExternal`
- `getModifiedSince`
- `getHistory`
- `blockNewRetrievalRequests`
- `allowRetrievalRequests`
- `getRetrievalStatistics`
- `resetRetrievalStatistics`
- `initCache`
- `eraseOldCacheEntries`
- `deleteCache`
- `setParameters`

– `changePassword`

All needed detailed information is in JavaDoc. So it will not be repeated here. Please note that some of those operations are planned for the next release. So they are not implemented yet but their *skeleton* is ready. Therefore those operations just return

```
598 Sorry.  This service is not implemented at the moment.
```

without causing an error.

## 9.9 Information for programmers

Here is a short summary for programmers. For detailed information please read the JavaDoc and see the source code.

### 9.9.1 WSDL

Web service in koLibRI is implemented following the *contract first* method. That means first all the types, messages, ports and bindings are defined in Web Services Description Language [17]. Those definitions of the WSDL file you can find in section 10.2 on page 54.

### 9.9.2 The generated Java package

Using those definitions, in WSDL file the needed java classes are generated with the help of Axis2 code generator plugin for Eclipse. There is also a command line version of code generator in Axis2 distribution. This machine generated code is for humans not easy to understand and therefore not in CVS. Instead in koLibRI just the compiled jar file is used and distributed. If you are interested in those classes you can generate them using the same `wsdl2java` tool and WSDL file.

### 9.9.3 The Server

On the server side, the java class `WebServiceServer` which implements `KolibriWebServiceSkeletonInterface` from generated jar accepts the SOAP messages and extracts the needed parameters to call the appropriate method in one of the middle layer classes `IngestLayer, AccessLayer, WSUtils`.

### 9.9.4 The Client

On the client side, the java class `WebServiceClient` which creates a new instance of `KolibriWebServiceStub` from generated jar takes the needed parameters and constructs the SOAP messages. It is designed as a helper class to make the communication on client side easier by accepting java data types instead of complex XML constructs.

### 9.9.5 Test Class

`TestWebServices` is on one side a simple example which demonstrates the usage of `WebServiceClient` and the communication with web service, on the other side it was necessary to test the functionality of `WebServiceServer`. It is not for production enviroments. You should implement similar calls to `WebServiceServer` (with or without using `WebServiceClient`) in your own system.

# 10 Appendix

## 10.1 The use of JHOVE in koLibRI

For the extraction of technical metadata, koLibRI uses the JSTOR/Harvard Object Validation Environment [8] (in short: JHOVE) version 1.1f (release of 01-08-2007) with some bugfixes which will be also contained in the upcoming maintenance release. JHOVE is used fully automatic by the ActionModule `MetadataGenerator` and its internal class `KopalHandler`, which works as an JHOVE `OutputHandler`. This handler enables the direct access to the created metadata within the program. `MetadataGenerator` is a customization of the `JhoveBase` class, which was matched and extended according to the given requirements.

JHOVE provides a very open module concept, that shall guarantee the support for future file formats. So in principal, JHOVE can create technical metadata for virtually any file format, preconditioned that an applying JHOVE module exists for it.

The used version of JHOVE supports the following file formats:

- AIFF-hul: Audio Interchange File Format
  - AIFF 1.3
  - AIFF-C
- ASCII-hul: ASCII-encoded text
  - ANSI X3.4-1986
  - ECMA-6
  - ISO 646:1991
- BYTESTREAM: Arbitrary bytestreams (always well-formed and valid)
- GIF-hul: Graphics Exchange Format (GIF)
  - GIF 87a
  - GIF 89a
- HTML-hul: Hypertext Markup Language (HTML)
  - HTML 3.2
  - HTML 4.0
  - HTML 4.01
  - XHTML 1.0 and 1.1

- JPEG-hul: Joint Photographic Experts Group (JPEG) raster images
  - JPEG (ISO/IEC 10918-1:1994)
  - JPEG File Interchange Format (JFIF) 1.2
  - Exif 2.0, 2.1 (JEIDA-49-1998), and 2.2 (JEITA CP-3451)
  - Still Picture Interchange File Format (SPIFF, ISO/IEC 10918-3:1997)
  - JPEG Tiled Image Pyramid (JTIP, ISO/IEC 10918-3:1997)
  - JPEG-LS (ISO/IEC 14495)
  - JPEG2000-hul: JPEG 2000
  - JP2 profile (ISO/IEC 15444-1:2000 / ITU-T Rec. T.800 (2000))
  - JPX profile (ISO/IEC 15444-2:2004)
- PDF-hul: Page Description Format (PDF)
  - PDF 1.0 through 1.6
    - Pre-press data exchange
    - PDF/X-1 (ISO 15930-1:2001)
    - PDF/X-1a (ISO 15930-4:2003)
    - PDF/X-2 (ISO 15390-5:2003)
    - PDF/X-3 (ISO 15930-6:2003)
  - Tagged PDF
  - Linearized PDF
  - PDF/A-1 (ISO/DIS 19005-1)
- TIFF-hul: Tagged Image File Format (TIFF) raster images
  - TIFF 4.0, 5.0, and 6.0
  - Baseline 6.0 Class B, G, P, and R
  - Extension Class Y
  - TIFF/IT (ISO 12639:2003)
    - File types CT, LW, HC, MP, BP, BL, and FP, and conformance levels P1 and P2
  - TIFF/EP (ISO 12234-2:2001)
  - Exif 2.0, 2.1 (JEIDA-49-1998), and 2.2 (JEITA CP-3451)
  - GeoTIFF 1.0
  - TIFF-FX (RFC 2301)
  - Profiles C, F, J, L, M, and S
  - Class F (RFC 2306)
  - RFC 1314
  - DNG (Adobe Digital Negative)

- UTF8-hul: UTF-8 encoded text
- WAVE: Audio for Windows
  - PCMWAVEFORMAT
  - WAVEFORMATEX
  - WAVEFORMATEXTENSION
  - Broadcast Wave Format (EBU N22-1997) version 0 and 1
- XML-hul: Extensible Markup Language (XML)
  - XML 1.0

In the context of the development of koLibRI, there have been developed two additional JHOVE modules, that are also included in the present koLibRI release. The first one is for *Disc Images according to ISO9660* (with support for Rock Ridge filename extensions), and a module for analyzing Postscript files.

The configuration of a JHOVE component is done with help of the JHOVE configuration file, whose path must be stated in the configuration file of koLibRI. In the current release, an example file is present within the `/config` directory and is named `jhove.conf`. Within this file, various entries for JHOVE modules exist, which look as followed:

```
<module>
  <class>de.langzeitarchivierung.kopal.jhove.Iso9660Module</class>
</module>
```

All stated modules are invoked sequential, until a module is found which applies to the file that is processed at the moment. The order of the modules is absolutely crutial here. Specialized modules should be placed above more generic ones. An example: An HTML file consists of a certain sequence of characters, but would be (correctly) recognized as a text file by the module for ASCII text, if it was invoked prior to the HTML module.

The PostScript module is able to verify a file using the Ghostscript [24] interpreter, if not configured it just tests for well-formedness. To use this verfication, please configure the path to the Ghostscript application as follows:

```
<module>
  <class>de.langzeitarchivierung.kopal.jhove.PsModule</class>
  <!--param>/usr/local/bin/gs</param-->
  <!--param>C:\\Programme\\gs\\gs8.54\\bin\\</param-->
</module>
```

The exact parameters of the Ghostscript call mustbe configured in the `PSModule` class.

To include the JHOVE functionalities in koLibRI, the JHOVE JAR files `jhove.jar`, `jhove-handler.jar` and `jhove-module.jar`, within the `/lib` directory of the koLibRI software, are used. These files can easily exchanged by the ones of upcoming maintenance releases of JHOVE, to profit from possible bugfixes and enhancements. Solely internal, structural changes within a new version of JHOVE could make possible changes to the source code of MetadataGenerator necessary.

## 10.2 WSDL file of the koLibRI Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="kolibri_web_services"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://kopal.langzeitarchivierung.de/ws/wsdl2java/"
targetNamespace="http://kopal.langzeitarchivierung.de/ws/wsdl2java/">

<wsdl:documentation>
/**
*  Web Services Description Language (WSDL) file for
*  kopal Library for Retrieval and Ingest (koLibRI) Web Services.
*
*  Please see the koLibRI documentation and source code for details.
*  koLibRI is free software under GNU Public License and sponsored by
*  Federal Republic of Germany, Federal Ministry of Education and Research
*
*  Copyright: Project kopal http://kopal.langzeitarchivierung.de
*  Author: Kadir Karaca Koçer, German National Library
*  June 2007
*/
</wsdl:documentation>

<!-- ***** T Y P E S ***** -->
<wsdl:types>
<xsd:schema elementFormDefault="qualified"
targetNamespace="http://kopal.langzeitarchivierung.de/ws/wsdl2java/">

<!-- Type USER -->
<xsd:complexType name="userType">
<xsd:sequence>
```

```
<xsd:element name="userName" type="xsd:token" />
<xsd:element name="password" type="xsd:token" />
<xsd:element name="institution" type="xsd:int" />
</xsd:sequence>
</xsd:complexType>

<!-- Type Custom XML -->
<xsd:complexType name="customXMLType">
<xsd:sequence>
<xsd:any namespace="##any" processContents="skip" />
</xsd:sequence>
</xsd:complexType>

<!-- Ingest -->
<xsd:element name="getFileFormatListElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="reloadFileFormatListElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<!-- Element OBJECT TO INGEST -->
<xsd:element name="ingestDocumentElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
<xsd:element name="policy" type="xsd:token" />
<xsd:element name="metadata" minOccurs="1" maxOccurs="6" type="tns:customXMLType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="ingestSIPElem">
<xsd:complexType>
<xsd:sequence>
```

```
<xsd:element name="user" type="tns:userType" />
<xsd:element name="urn" type="xsd:token" />
<xsd:element name="linkToSIP" type="xsd:token" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="blockNewIngestRequestsElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="allowIngestRequestsElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="getIngestStatisticsElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="resetIngestStatisticsElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>


<!-- OBJECT TO RETRIEVE -->
<xsd:element name="getMETSElem">
<xsd:complexType>
<xsd:sequence>
```

```
<xsd:element name="user" type="tns:userType" />
<xsd:element name="objectId" type="xsd:token" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="getFileElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
<xsd:element name="objectId" type="xsd:token" />
<xsd:element name="fileName" type="xsd:token" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="getDIPElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
<xsd:element name="objectId" type="xsd:token" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="getDIPExternalElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
<xsd:element name="objectId" type="xsd:token" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="getModifiedSinceElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
<xsd:element name="modificationdate" type="xsd:dateTime" nillable="true"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="getHistoryElem">
```

```
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
<xsd:element name="objectId" type="xsd:token" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="blockNewRetrievalRequestsElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="allowRetrievalRequestsElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="getRetrievalStatisticsElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="resetRetrievalStatisticsElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="initCacheElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
```

```
<xsd:element name="rootdir" type="xsd:token" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="eraseOldCacheEntriesElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
<xsd:element name="timelimit" type="xsd:int" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="deleteCacheElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<!-- Change Password -->
<xsd:element name="changePasswordElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
<xsd:element name="newPassword" type="xsd:token" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<!-- Parameters -->
<xsd:element name="setParametersElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="user" type="tns:userType" />
<xsd:element name="diasRetrieverURL" type="xsd:token" minOccurs="0" maxOccurs="1" />
<xsd:element name="diasLoaderURL" type="xsd:token" minOccurs="0" maxOccurs="1" />
<xsd:element name="fileFormatListURL" type="xsd:token" minOccurs="0" maxOccurs="1" />
<xsd:element name="cacheRootDir" type="xsd:token" minOccurs="0" maxOccurs="1" />
<xsd:element name="protocolRetrievalWS" type="xsd:token" minOccurs="0" maxOccurs="1" />
<xsd:element name="urlRetrievalWS" type="xsd:token" minOccurs="0" maxOccurs="1" />
<xsd:element name="tempDir" type="xsd:token" minOccurs="0" maxOccurs="1" />
```

```
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<!-- Type RESPONSE -->
<xsd:element name="responseElem">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="statusCode" type="xsd:int" />
<xsd:element name="responseText" type="xsd:string" />
<xsd:element name="diasId" type="xsd:string" minOccurs="0" maxOccurs="1" />
<xsd:element name="waitTime" type="xsd:int"  minOccurs="0" maxOccurs="1" />
<xsd:element name="linkToFile" type="xsd:string" minOccurs="0" maxOccurs="1" />
<xsd:element name="extraInfo" type="xsd:string" minOccurs="0" maxOccurs="1" />
<xsd:element name="errorMessage" type="xsd:string" minOccurs="0" maxOccurs="1" />
<xsd:element name="customdata" type="tns:customXMLType" minOccurs="0" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</wsdl:types>


<!-- ***** M E S S A G E S ***** -->
<!-- Ingest -->
<wsdl:message name="getFileFormatListMsg">
<wsdl:part name="request" element="tns:getFileFormatListElem" />
</wsdl:message>

<wsdl:message name="reloadFileFormatListMsg">
<wsdl:part name="request" element="tns:reloadFileFormatListElem" />
</wsdl:message>

<wsdl:message name="ingestDocumentMsg">
<wsdl:part name="request" element="tns:ingestDocumentElem" />
</wsdl:message>

<wsdl:message name="ingestSIPMsg">
<wsdl:part name="request" element="tns:ingestSIPElem" />
</wsdl:message>

<wsdl:message name="blockNewIngestRequestsMsg">
<wsdl:part name="request" element="tns:blockNewIngestRequestsElem" />
</wsdl:message>
```

```
<wsdl:message name="allowIngestRequestsMsg">
<wsdl:part name="request" element="tns:allowIngestRequestsElem" />
</wsdl:message>

<wsdl:message name="getIngestStatisticsMsg">
<wsdl:part name="request" element="tns:getIngestStatisticsElem" />
</wsdl:message>

<wsdl:message name="resetIngestStatisticsMsg">
<wsdl:part name="request" element="tns:resetIngestStatisticsElem" />
</wsdl:message>

<!-- Retrieval -->
<wsdl:message name="getMETSMsg">
<wsdl:part name="request" element="tns:getMETSElem" />
</wsdl:message>

<wsdl:message name="getFileMsg">
<wsdl:part name="request" element="tns:getFileElem" />
</wsdl:message>

<wsdl:message name="getDIPMsg">
<wsdl:part name="request" element="tns:getDIPElem" />
</wsdl:message>

<wsdl:message name="getDIPExternalMsg">
<wsdl:part name="request" element="tns:getDIPExternalElem" />
</wsdl:message>

<wsdl:message name="getModifiedSinceMsg">
<wsdl:part name="request" element="tns:getModifiedSinceElem" />
</wsdl:message>

<wsdl:message name="getHistoryMsg">
<wsdl:part name="request" element="tns:getHistoryElem" />
</wsdl:message>

<wsdl:message name="blockNewRetrievalRequestsMsg">
<wsdl:part name="request" element="tns:blockNewRetrievalRequestsElem" />
</wsdl:message>

<wsdl:message name="allowRetrievalRequestsMsg">
<wsdl:part name="request" element="tns:allowRetrievalRequestsElem" />
```

```
</wsdl:message>

<wsdl:message name="getRetrievalStatisticsMsg">
<wsdl:part name="request" element="tns:getRetrievalStatisticsElem" />
</wsdl:message>

<wsdl:message name="resetRetrievalStatisticsMsg">
<wsdl:part name="request" element="tns:resetRetrievalStatisticsElem" />
</wsdl:message>

<wsdl:message name="initCacheMsg">
<wsdl:part name="request" element="tns:initCacheElem" />
</wsdl:message>

<wsdl:message name="eraseOldCacheEntriesMsg">
<wsdl:part name="request" element="tns:eraseOldCacheEntriesElem" />
</wsdl:message>

<wsdl:message name="deleteCacheMsg">
<wsdl:part name="request" element="tns:deleteCacheElem" />
</wsdl:message>

<!-- Change password -->
<wsdl:message name="changePasswordMsg">
<wsdl:part name="request" element="tns:changePasswordElem" />
</wsdl:message>

<!-- setParameters -->
<wsdl:message name="setParametersMsg">
<wsdl:part name="request" element="tns:setParametersElem" />
</wsdl:message>

<!-- Response -->
<wsdl:message name="diasResponse">
<wsdl:part name="response" element="tns:responseElem" />
</wsdl:message>


<!-- ***** P O R T S ***** -->
<wsdl:portType name="KolibriServiceSOAP">
<!-- Ingest -->
<wsdl:operation name="getFileFormatList">
<wsdl:input message="tns:getFileFormatListMsg" />
<wsdl:output message="tns:diasResponse" />
```

```
</wsdl:operation>

<wsdl:operation name="reloadFileFormatList">
<wsdl:input message="tns:reloadFileFormatListMsg" />
<wsdl:output message="tns:diasResponse" />
</wsdl:operation>

<wsdl:operation name="ingestDocument">
<wsdl:input message="tns:ingestDocumentMsg" />
<wsdl:output message="tns:diasResponse" />
</wsdl:operation>

<wsdl:operation name="ingestSIP">
<wsdl:input message="tns:ingestSIPMsg" />
<wsdl:output message="tns:diasResponse" />
</wsdl:operation>

<wsdl:operation name="blockNewIngestRequests">
<wsdl:input message="tns:blockNewIngestRequestsMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="allowIngestRequests">
<wsdl:input message="tns:allowIngestRequestsMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="getIngestStatistics">
<wsdl:input message="tns:getIngestStatisticsMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="resetIngestStatistics">
<wsdl:input message="tns:resetIngestStatisticsMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<!-- Retrieval -->
<wsdl:operation name="getMETS">
<wsdl:input message="tns:getMETSMsg" />
<wsdl:output message="tns:diasResponse" />
</wsdl:operation>

<wsdl:operation name="getFile">
```

```
<wsdl:input message="tns:getFileMsg" />
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="getDIP">
<wsdl:input message="tns:getDIPMsg" />
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="getDIPExternal">
<wsdl:input message="tns:getDIPExternalMsg" />
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="getModifiedSince">
<wsdl:input message="tns:getModifiedSinceMsg" />
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="getHistory">
<wsdl:input message="tns:getHistoryMsg" />
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="blockNewRetrievalRequests">
<wsdl:input message="tns:blockNewRetrievalRequestsMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="allowRetrievalRequests">
<wsdl:input message="tns:allowRetrievalRequestsMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="getRetrievalStatistics">
<wsdl:input message="tns:getRetrievalStatisticsMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="resetRetrievalStatistics">
<wsdl:input message="tns:resetRetrievalStatisticsMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>
```

```
<wsdl:operation name="initCache">
<wsdl:input message="tns:initCacheMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="eraseOldCacheEntries">
<wsdl:input message="tns:eraseOldCacheEntriesMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<wsdl:operation name="deleteCache">
<wsdl:input message="tns:deleteCacheMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>

<!-- Administration -->
<wsdl:operation name="setParameters">
<wsdl:input message="tns:setParametersMsg" />
<wsdl:output message="tns:diasResponse" />
</wsdl:operation>

<wsdl:operation name="changePassword">
<wsdl:input message="tns:changePasswordMsg"/>
<wsdl:output message="tns:diasResponse"/>
</wsdl:operation>
</wsdl:portType>


<!-- ***** B I N D I N G S  ***** -->
<wsdl:binding name="bindingKolibri" type="tns:KolibriServiceSOAP">
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />

<wsdl:operation name="getFileFormatList">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#getFileFormatList" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="reloadFileFormatList">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#reloadFileFormatList" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>
```

```
<wsdl:operation name="ingestDocument">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#ingestDocument" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="ingestSIP">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#ingestSIP" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="blockNewIngestRequests">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#blockNewIngestRequests" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="allowIngestRequests">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#allowIngestRequests" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="getIngestStatistics">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#getIngestStatistics" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="resetIngestStatistics">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#resetIngestStatistics" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<!-- Retrieval -->
<wsdl:operation name="getMETS">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#getMETS" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="getFile">
```

```
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#getFile" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="getDIP">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#getDIP" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="getDIPExternal">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#getDIPExternal" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="getModifiedSince">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#getModifiedSince" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="getHistory">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#getHistory" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="blockNewRetrievalRequests">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#blockNewRetrievalRequests" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="allowRetrievalRequests">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#allowRetrievalRequests" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal"/></wsdl:output>
</wsdl:operation>

<wsdl:operation name="getRetrievalStatistics">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#getRetrievalStatistics" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
```

```
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="resetRetrievalStatistics">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#resetRetrievalStatistics" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="initCache">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#initCache" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="eraseOldCacheEntries">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#eraseOldCacheEntries" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="deleteCache">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#deleteCache" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="setParameters">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#setParameters" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>

<wsdl:operation name="changePassword">
<soap:operation soapAction="http://bernstein.d-nb.de/kolibri#changePassword" />
<wsdl:input><soap:body use="literal" /></wsdl:input>
<wsdl:output><soap:body use="literal" /></wsdl:output>
</wsdl:operation>
</wsdl:binding>


<!-- ***** S E R V I C E S  ***** -->
<wsdl:service name="kolibriWebService">
<wsdl:port name="kolibri_ws" binding="tns:bindingKolibri">
```

```
<soap:address location="http://bernstein.d-nb.de/kolibri/" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

## 10.3 Direct usage of the interfaces of DIAS

For the direct use of the access and ingest interfaces of DIAS, the commandline tools `DiasAccess` and `DiasIngest` can be used. They are realized through `main` methods within the classes `retrieval.DiasAccess` and `ingest.DiasIngest`. Knowledge of the DIP and SIP interface specifications [10] [11] is preconditioned.

It has to be stated here again, that, as for the WorkflowTool also for these tools, the configuration of a keystore file is typically necessary, as the access to DIAS should be realized over an encrypted connection. A `known hosts` file also has to be stated correctly.

### DiasIngest

```
java -jar DiasIngest.jar -h:
  -hp, --show-properties    Print the system properties and continue.
   -a, --address            The server address.
   -f, --file               The file to submit.
   -h, --help               Print this dialog and exit.
   -n, --port               The server port.
   -p, --password           The CMPassword of the CMUser.
   -t, --testdias           Print dias responses.
   -u, --user               The submitting CMUser.
```

### DiasAccess

```
java -jar DiasAccess.jar -h:
   -d, --dip-format         The requested dip format [zip|tar|tar.gz].
   -r, --response-format    The requested response format [xml|html].
   -x, --ext-id             The requested external id.
   -i, --int-id             The requested internal id.
  -hp, --show-properties    Print the system properties and continue.
   -t, --request-type       The type of request [metadata|fullmetadata|asset].
   -a, --address            The server address.
   -f, --file               Parse a file as a Dias response and prints it.
   -g, --download           If no file is specified the file in the dias response is
                              downloaded and saved in the current directory. Else the
                              specified file is downloaded.
   -h, --help               Print this dialog and exit.
```

```
-l, --list              Returns a list of all metadata sets for an external id.
-n, --port              The server port.
-p, --print             Prints the dias response.
-u, --unsecure          Use unsecure access to dias.
```

## 10.4 The TIFF Image Metadata Processor

The TiffIMP is useful to validate TIFF images and process their header metadata. Use it to show, validate and repair the TIFF header metadata. It is using the JHOVE to validate the TIFF images and to be used via API or as a commandline tool.

### TiffIMP as a commandline tool

```
usage: tiffimp [options]
    -h, --help                      Very surprising: A help message :-)
    -r, --rewrite-tiff-header       The TIFF header metadata will be rewritten. Please
                                      notice that only the following tasks are able to be
                                      repaired yet:
                                      (a) Word-alignedness of all TIFF ASCII tags.
                                      (b) PageNumber 297 0x0129 set to the valid
                                      TIFF_SHORT count of 2 bytes.
    -i, --input-filename <file>     The filename of the TIFF image to process.
    -o, --output-filename <file>    The filename of the corrected TIFF image. Default is
                                      _cor.tif.
    -j, --jhove-xml-output          Outputs the Jhove XML output using the TIFF-hul module.
    -c, --validate-tiff-image       Validates the TIFF image according to TIFF specification
                                      compliance using the Jhove
                                      (see http://hul.harvard.edu/jhove).
    -s, --show-header-metadata      Outputs the TIFF image header metadata tags and their
                                      content to standard out. This is the default option.
    -f, --force-overwrite           Forces the input image file to be overwritten if header
                                      metadata shall be corrected.
    -n, --rewrite-if-not-valid      Only rewrites the TIFF header if the image file is not
                                      wellformed or not valid according to the Jhove.
    -q, --quiet                     Run tiffimp in quiet mode.
    -v, --verbose                   Run tiffimp in verbose mode.
```

## 10.5 Errorcodes at System.exit

### Miscellaneous

(0) Regular ending of the program

### opal.WorkflowTool, kopal.retrieval.DiasAccess

(1) Commandline and its arguments are not correct
Configuration invalid for WorkflowTool
Configuration invalid for ProcessStarter.

(2) Could not load and initialize process starters. Please check the commandline flag -p or the DefaultProcessStarter field in the main config file.

(3) Programm was terminated and there was an error while processing some of the lists elements: Not everything has finished correctly! Please check the logfile.

(9) Could not create logfile.

(12) Database initialization failed.

### kopal.processstarter.MonitorHotfolderBase

(6) The given hotfolder path does not exist or is not a directory.

(14) Error processing current File. No file or no directory.

### kopal.Policy

(7) Configuration file could not be parsed: Exception while parsing the policy file.

### kopal.util.FormatRegistry.java

(10) No backup file for format registry found! Formats can not be identified.

(13) Error parsing the DIAS format registry backup file.

(16) Error accessing the format registry backup file.

### kopal.util.kopalXMLParser, kopal.util.HTMLUtils

(11) Parse error in XML file, Parse error in XML string.

### kopal.processstarter.MonitorHttpLocationBase

(15) Error accessing URL

## 10.6 Error handling und loglevel

`SEVERE` The program has to be terminated because of an severe failure. *Example*: A configuration file cannot be found.

`WARNING` Warnings are logged for failures that do not require the termination of the program. *Example*: The processing of a list element is stopped, because the module set the status `ERROR`. The next module will be processed.

`INFO` All information, relevant to the user, are logged within `INFO`. *Example*: The start of a Server, successful parsing of a configuration file, addition of an element to the ProcessQueue, etc.

`FINE` `FINE` loggs all informations which have little relevance to the user and are only interesting for debugging purposes. *Example*: Internal messages of the list processing method `WorkflowTool.process()`.

`FINER/FINEST` With `FINER` and `FINEST`, the finest debug messages can be logged, which are seldom needed, even for debugging. *Example*: Messages about `notify()` and `wait()` for the debugging of threads.

`ALL/OFF` All messages, respectively no messages, are logged.

# References

[1] DIAS (Digital Information Archiving System)
http://www-5.ibm.com/nl/dias/

[2] Reference Model for an Open Archival Information System (OAIS)
http://ssdoo.gsfc.nasa.gov/nost/isoas/ref_model.html

[3] Uniform Resource Name
http://www.persistent-identifier.de

[4] METS (Metadata Encoding & Transmission Standard)
http://www.loc.gov/standards/mets/

[5] LMER (Long-term preservation Metadata for Electronic Resources)
http://d-nb.de/standards/lmer/lmer.htm

[6] kopal – Co-operative Development of a Long-Term Digital Information Archive
http://kopal.langzeitarchivierung.de/

[7] nestor – Network of Expertise in Long-Term Storage of Digital Resources
http://www.langzeitarchivierung.de

[8] JHOVE (JSTOR/Harvard Object Validation Environment)
http://hul.harvard.edu/jhove/

[9] The Free Software Foundation
http://www.fsf.org

[10] DIAS SIP Interface Specification
http://kopal.langzeitarchivierung.de/downloads/kopal_DIAS_SIP_Interface_Specification.pdf

[11] DIAS DIP Interface Specification
http://kopal.langzeitarchivierung.de/downloads/kopal_DIAS_DIP_Interface_Specification.pdf

[12] Hibernate – Relational Persistence for Java
http://www.hibernate.org

[13] SOAP (Simple Object Access Protocol)
http://www.w3.org/TR/soap/

[14] AXIS2
http://ws.apache.org/axis2/

[15] Apache Tomcat
http://tomcat.apache.org

[16] Dublin Core Metadata Initiative
http://dublincore.org

[17] WSDL (Web Service Description Language)
http://www.w3.org/TR/wsdl/

[18] German National Library
http://www.d-nb.de/eng/

[19] Goettingen State and University Library
http://www.sub.uni-goettingen.de

[20] IBM Deutschland GmbH
http://www.ibm.com/de/

[21] Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen
http://www.gwdg.de

[22] Universal Object Format – An archiving and exchange format for digital objects
http://kopal.langzeitarchivierung.de/downloads/kopal_Universelles_Objektformat.pdf

[23] Apache XMLBeans
http://xmlbeans.apache.org/

[24] Ghostscript
http://www.ghostscript.com/awki

[25] Common Query Language (CQL)
http://www.loc.gov/standards/sru/cql/

[26] Global Digital Format Registry (GDFR)
http://hul.harvard.edu/gdfr/

[27] PRONOM – The Online Registry of Technical Information
http://www.nationalarchives.gov.uk/pronom/