

# **Cortex Documentation**

Fraunhofer-Institute for Intelligent Analysis- and Information Systems IAIS

November 18, 2011



---

Wir weisen darauf hin, dass dieses Dokument urheberrechtlich geschützte Inhalte und Know-How enthält, die Betriebsgeheimnisse des Fraunhofer- Instituts IAIS darstellen. Das Dokument darf nur zur Information für das Cortex-System verwendet werden. Jede Verwendung für andere Zwecke, Vervielfältigung, Modifikation und Weitergabe an Dritte ohne vorherige schriftliche Zustimmung von Fraunhofer IAIS verletzt die Rechte der Fraunhofer Gesellschaft und ist Gegenstand von Unterlassungs- und Schadensersatzansprüchen.

Contact: [kai.stalman@iais.fraunhofer.de](mailto:kai.stalman@iais.fraunhofer.de)



# Contents

<b>1</b>	<b>Installation and Setup</b>	<b>1</b>
1.1	Prerequisites . . . . .	1
1.1.1	For compilation . . . . .	1
1.1.2	For running a precompiled System . . . . .	1
1.2	Start from Eclipse . . . . .	1
1.3	Creating a Cortex Binary Distribution . . . . .	2
1.4	Installation of a Cortex Binary Distribution . . . . .	2
1.5	All-in-one Demo Installation and Start . . . . .	3
1.5.1	Installation and Start on a single machine . . . . .	3
1.5.2	ASC Installation and Start . . . . .	3
1.5.3	Installation and Start on a multi-machine setup . . . . .	3
1.5.4	Possible problems on start up . . . . .	4
1.6	Installation . . . . .	4
1.7	Operating . . . . .	5
<b>2</b>	<b>REST Endpoints</b>	<b>7</b>
2.1	Data Access . . . . .	7
2.1.1	List Single Items . . . . .	7
2.1.2	State change . . . . .	8
2.1.3	Listing All Components . . . . .	8
2.1.4	Listing Single Components . . . . .	9
2.1.5	Listing Subcomponents . . . . .	10
2.2	User Profiles . . . . .	10
2.2.1	Application: Register . . . . .	10
2.2.2	Application: Register . . . . .	11
2.2.3	Topics: Add . . . . .	11
2.2.4	Topics: Get . . . . .	11
2.2.5	Topics: Remove . . . . .	12
2.2.6	Journeys: Get head . . . . .	12
2.2.7	Journeys: Deactivate lock . . . . .	13
2.2.8	Journeys: Activate lock . . . . .	13

2.2.9	Journeys: Is lock activated . . . . .	14
2.2.10	Journeys: Add Update . . . . .	14
2.2.11	Journeys: Delete Update . . . . .	14
2.2.12	Journeys: Add Named Journey . . . . .	15
2.2.13	Journeys: Rename Named Journey . . . . .	15
2.2.14	Journeys: Delete Named Journey . . . . .	16
2.2.15	Journeys: Get Named Journey . . . . .	16
2.2.16	Journeys: Get Data . . . . .	17
2.2.17	Journeys: Erase Data . . . . .	17
2.2.18	Journeys: Get Update for timestamp . . . . .	17
2.2.19	Journeys: Get Updates between timestamp . . . . .	18
2.2.20	Journeys: Store Journey Data . . . . .	18
2.2.21	Journeys: Get timestamps . . . . .	19
2.2.22	Journeys: Get current state for a topic . . . . .	19
2.2.23	Journeys: Get all current states . . . . .	20
2.2.24	Journeys: Get all current state timestamps . . . . .	20
2.2.25	Journeys: Get all breadcrumbs . . . . .	20
2.2.26	Journeys: Get latest breadcrumbs . . . . .	21
2.2.27	Collections: Get collection names . . . . .	21
2.2.28	Collections: Delete collection . . . . .	22
2.2.29	Collections: Add item to collection . . . . .	22
2.2.30	Collections: Delete from collection . . . . .	22
2.2.31	Collections: Has item in collection . . . . .	23
2.2.32	Collections: Get collection . . . . .	23
2.2.33	Collections: Get collection size . . . . .	24
2.2.34	Collections: Get collection's items from start position on . . . . .	24
2.2.35	Collections: Store collection data . . . . .	25
2.2.36	Configurations: Add item to configuration . . . . .	25
2.2.37	Configurations: Delete item from configuration . . . . .	26
2.2.38	Configurations: Has item in configuration . . . . .	26
2.2.39	Configurations: Get configuration . . . . .	26
2.2.40	Configurations: Store configuration . . . . .	27
2.2.41	Services: Add to Service . . . . .	27
2.2.42	Services: Delete from Service . . . . .	28
2.2.43	Services: Has item in Service . . . . .	28
2.2.44	Services: Get Service . . . . .	28
2.2.45	Services: Store Service . . . . .	29

2.3	Search and Facets . . . . .	29
2.3.1	Normal Search . . . . .	29
2.3.2	Cluster Search . . . . .	31
2.4	Ingest . . . . .	32
2.4.1	Ingest A SIP . . . . .	32
2.5	Reporting Interface . . . . .	32
2.5.1	Listing A Single Report . . . . .	33
2.5.2	Listing All Reports . . . . .	34
2.6	Hopping . . . . .	36
2.6.1	Retrieving first order relations . . . . .	36
2.6.2	Retrieving preconfigured first order relations . . . . .	36
2.6.3	Fuzzy Retrieve all destinations of a given type for a given entity using the search index. . . . .	37
<b>3</b>	<b>Augmented SIP Creator (ASC)</b>	<b>39</b>
3.1	Introduction . . . . .	39
3.2	Building the ASC . . . . .	39
3.3	Directory structure for storing metadata . . . . .	39
3.3.1	Directory structure defaults . . . . .	39
3.4	Controlling ASC . . . . .	40
3.5	myCortex . . . . .	43
3.5.1	Start-Up . . . . .	44
3.5.2	Working with myCortex . . . . .	44
<b>4</b>	<b>Monitoring</b>	<b>47</b>
4.1	Overview . . . . .	47
4.2	AdminServer . . . . .	47
4.2.1	Servers overview . . . . .	47
4.2.2	Server details . . . . .	48
4.2.3	Services . . . . .	48
4.2.4	Service summary . . . . .	48
4.2.5	Service details . . . . .	49
4.2.6	Thread dump . . . . .	52
4.2.7	Profiling . . . . .	52
4.2.8	Solr . . . . .	52
4.2.9	Settings . . . . .	52
4.3	ICortexServiceMonitor . . . . .	53
4.4	Communication between the AdminServer and ICortexServiceMonitor . . . . .	54

4.5	REST Endpoints . . . . .	54
4.5.1	Cluster members . . . . .	54
4.5.2	Registered servers . . . . .	55
4.5.3	Monitoring data . . . . .	56
<b>5</b>	<b>TestSuiteGenerator</b>	<b>59</b>
5.1	Problem . . . . .	59
5.2	To-Be-concept . . . . .	59
5.3	Implementation . . . . .	60
5.3.1	TestSet.class . . . . .	60
5.3.2	TestSuiteGenerator.class . . . . .	61
5.4	Usage . . . . .	61
5.4.1	AllTestsCreator.class . . . . .	61
5.4.2	HierarchyTests.class . . . . .	62
5.5	Problems . . . . .	62
5.5.1	Java reflections . . . . .	62
5.5.2	Automatic execution after generating the test suites . . . . .	62
<b>6</b>	<b>Native Content</b>	<b>63</b>
6.1	Mode of Operation . . . . .	63
<b>7</b>	<b>Quality Assurance of Java Code in the Cortex Platform</b>	<b>65</b>
7.1	Status . . . . .	65
7.2	Introduction . . . . .	65
7.3	Goals and category groups for quality ensuring measurements . . . . .	65
7.4	Organization of the Eclipse Workspace . . . . .	66
7.4.1	Required Eclipse Plugins . . . . .	66
7.4.2	Settings in Eclipse . . . . .	68
7.5	Organization of the projects in the Eclipse-workspace . . . . .	72
7.6	Using SVN . . . . .	73
7.7	Test-driven development . . . . .	74
7.7.1	Purpose . . . . .	74
7.7.2	approach . . . . .	74
7.7.3	Types of Unit-tests . . . . .	75
7.8	DBC: Design By Contract . . . . .	76
7.8.1	Contracts or Agreements . . . . .	77
7.8.2	Handling exceptions . . . . .	78
7.9	Logging in production-, and testsystems . . . . .	79
7.9.1	Purpose . . . . .	79

---

7.9.2	Slf4j as a façade for logging . . . . .	80
7.9.3	Configuration . . . . .	81
7.9.4	Declaring loggers in the source-code . . . . .	82
7.9.5	utilizing Loggers, Best Practices . . . . .	82
7.10	CI: Continous Integration with Hudson and Sonar . . . . .	83
7.11	Sonar for examination of rule violations . . . . .	84
7.11.1	The Sonar rulesets . . . . .	84
7.11.2	Rule violations in the Cortex code . . . . .	84
7.12	Current rules for the small scale quality checks . . . . .	85
7.13	Declaration of the Sonar rules . . . . .	86
<b>8</b>	<b>Software performance testing</b>	<b>105</b>
8.1	Goals . . . . .	105
8.2	Definitions . . . . .	105
8.2.1	Load tests . . . . .	105
8.2.2	Performance tests . . . . .	106
8.2.3	Stress tests . . . . .	106
8.2.4	Failover tests . . . . .	106
8.3	JMeter . . . . .	107
8.4	Test stories . . . . .	107
8.4.1	synthetic tests . . . . .	107
8.4.2	simulations . . . . .	108
8.4.3	journeys . . . . .	109
8.4.4	replicating real users . . . . .	109
8.5	Test implementations . . . . .	109
8.5.1	Problems and solutions . . . . .	109
8.5.2	Concrete test implementations . . . . .	110
8.5.3	Combined tests . . . . .	110
8.6	Results . . . . .	110
<b>9</b>	<b>Integration Tests</b>	<b>111</b>
9.1	Introduction . . . . .	111
9.2	Full-Cortex-Test . . . . .	111
9.2.1	Architecture . . . . .	111
9.2.2	Design and application flow . . . . .	112
9.3	Functional tests . . . . .	112
9.3.1	Ingest validation . . . . .	112
9.3.2	Rest-Interface-Test . . . . .	112

9.4	Known limitations / Problems / TODOs . . . . .	113
9.4.1	MQ-Ingest finish notification . . . . .	113
9.4.2	Report-handling . . . . .	113
9.4.3	Starter . . . . .	113

# 1 Installation and Setup

## 1.1 Prerequisites

### 1.1.1 For compilation

Java 6, Eclipse 3.6+ and SVN and Maven-Plugins are required.

### 1.1.2 For running a precompiled System

Java 6

## 1.2 Start from Eclipse

Cortex can be built and started as a normal Java application out of Eclipse. You can use Eclipse 3.6 or Eclipse 3.7 for checkout and compilation. Please make sure an up-to-date Maven plugin which is not in beta-phase is installed. Any Eclipse integrated stable SVN client will work.

To get the Cortex, you have to check it out from the development SVN. Please ask for login credentials. You can checkout the current version out of the trunk of the development-SVN. The latest tested release is available in the "tags" directory of the SVN under the directory with the current release version e.g. "R2/110805" for Release 2 and August 5th, 2011.

In Eclipse, activate "Project/Build Automatically".

If your computer is part of a LAN with several simultaneously running instances of Cortex you may encounter lags or even some exceptions on startup. For an explanation and some work around of this problem please take a look at "Possible problems on start up"[1.5.4].

**Starting** In the project Cortex the starter is located in the package `de.fhg.iais.cortex`.

The starter provides a main method to run all servers with factory defaults.

The configurations, especially the paths to the storage, the SOLR index, as well as the ftp defaults are configured in the file `Cortex/conf/cortex.properties`. We assume this file is self-explanatory.

You can checked the started server at `http://localhost:8080`

**Ingest** For the pre-ingest process and ingest, please have a running **Cortex-server** on your system. Then you can start the ASC (in package `de.fhg.iais.asc`). The ASC properties also can be configured in `ASC/asc.properties` and `ASC/provider.properties`. These files are also self-explanatory.

If you prefer a graphical user interface to run this process you will find the ASC UI in the file `ASCLight` in the package `de.fhg.iais.asc.ui` of the `ASC_UI` project.

The ASC harvests data, transforms them and sends them to the cortex core where the data is being cross-linked, indexed and stored.

### 1.3 Creating a Cortex Binary Distribution

Cortex-Build requires Maven, so you need an up-to-date Maven version. The build process can be started on commandline.

1. Check out all Cortex projects from SVN (ask for repository access). Only once (*or after deleting the maven repository*), run `mvn install` in `Cortex_InstallExternalJARs`.
2. In the project `Cortex_build`, you find a file called `makeall.sh` (*for Linux and Mac*) and a `makeall.bat` (*for Windows*). Start the script corresponding to your system. On Mac/Linux you need to make the file executable first by executing `chmod +x makeall.sh`.  
In case the build process does not complete without errors, please report the error messages to our development team.
3. After the system has been build successfully, the directory `Cortex_Build/target` contains the files `cortexassembly.zip` and `demo.zip`. The archive and directory contain all packages needed for the installation.

### 1.4 Installation of a Cortex Binary Distribution

The Cortex binary distribution within the file `cortexassembly.zip` contains the following packages:

**asc:** the complete Augmented SIP Creator for harvesting, transformation and uploading the data

**core:** single machine version of the Cortex Core including webservers

**core4:** packages for a distributed installation

**tools:** maintenance tools, so far a tool for inspection of the storage

**webserver:** separate webserver for distributed installation of Cortex

## 1.5 All-in-one Demo Installation and Start

1. Unzip the contents of `Cortex_Build/target/demo.zip` into a directory.
2. Navigate to the directory containing `cortex-starter.jar`, `asc.jar` and several other directories and files.
3. If required, change the files `asc.properties` and `provider.properties` for the ASC and `/conf/cortex.properties` for the core.
4. Start Cortex by executing `java -jar cortex-starter.jar`
5. Start the ASC by executing `asc.bat` or `asc.sh` (*remember to make executable before*)

### 1.5.1 Installation and Start on a single machine

1. Copy the contents from `cortexassembly/core` into a directory.
2. Navigate to the directory containing `cortex-starter.jar`, `asc.jar` and several other directories and files.
3. If required, change `/conf/cortex.properties` for the core.
4. Start Cortex by executing `java -jar cortex-starter.jar`

### 1.5.2 ASC Installation and Start

1. Copy the contents from `cortexassembly/asc` into a directory.
2. Browse to the directory containing `asc.sh`, `asc.bat`, `asc.jar` and several other directories and files.
3. Adjust the parameters server address, local path, general behaviour in `asc.properties` and harvesting, provider information, process control in `provider.properties`.
4. Start ASC: `asc.bat` or `asc.sh` (*remember to make the script executable before*)

### 1.5.3 Installation and Start on a multi-machine setup

The content of `cortexassembly/core4` contains the following components:

**ArchiveServer:** Filesystem-based storage containing the resources (metadata and binaries).  
Can be installed on a separate machine.

**CSServer:** Cortex-middleware containing a centralized REST Interface for all clients. Can be installed on one or more machines.

**FTPServer:** Receives Binaries. Can be installed on a separate machine.

**IndexServer:** Contains the SOLR Server.

**TripleStoreServer:** Contains the triplestore, currently Jena via Sparql Endpoint to Derby DB.

### 1.5.4 Possible problems on start up

You may encounter time lags or even exceptions when trying to start Cortex, especially when several computers connected to the same LAN are running Cortex simultaneously (which may be the case in development or test environments). One possible source of such problems is Hazelcast, a data grid framework used by Cortex for cross server communications. Hazelcast uses per default a multicast protocol for data transmissions between machines belonging to the same Hazelcast group. If your LAN doesn't support multicasting or one or more machines in your Hazelcast group produce high network latency you should consider changing Hazelcasts default configuration.

Hazelcast's configuration can be found in `conf/hazelcast/default.xml`. There are a few settings you may need to change.

**Hazelcast group** The content of the `<name>` child of the `<group>` node works as Hazelcast's group identifier. To make sure your machine is the only one in a specific Hazelcast group the name should be changed to an unique one.

**Communication protocol** Besides multicasting Hazelcast supports TCP/IP as its transport protocol. Switching from multicast to TCP/IP or vice versa can be achieved by setting the `enabled` attributes of the `<multicast>` and `<tcp-ip>` nodes to the appropriate boolean value.

**Ports** By default Hazelcast uses ports starting at 5701 for administrative purposes and in case of multicasting port 54327 for data transfer. The first machine registering to a specific Hazelcast group listens on port 5701, the next one on port 5702 and so on. The starting port is configured in the `<port>` node. If for some reason the multicast port 54327 is already in use it may be changed in the `<multicast-port>` node.

## 1.6 Installation

Installation works similar to "Installation and Start on a single machine"[1.5.1] by simply copying the files. In case of a distributed installation the components are being copied to the various machines or VM's.

## **1.7 Operating**

All components are started by a runnable JAR, i.e. `java -jar <component>.jar`. You should consider to assign more memory to the Java VM, i.e. `java -Xms1G -Xmx2G -jar <component>.jar`. All components are configured by a property file: `/conf/cortex.properties` where the machine's address and ports have to be configured. This is essential for the interaction of the distributed components.



## 2 REST Endpoints

This chapter is listing the service endpoints of the REST interface. This REST interface can be called by an arbitrary User Interface. The REST interface exposed on port 9998 but its port can be changed within the properties-file `cortex.properties`.

### 2.1 Data Access

Access items / components. Client must set identifier to valueOf `CURRENT_IDENTIFIER` to retrieve the identifier from the stateServer. For setting the identifier in the stateServer use `POST setItem`. Getter do not change the state!

#### 2.1.1 List Single Items

Get Id from stateserver if `CURRENT_IDENTIFIER` was set.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/access/{identifier}		APPLICATION-JSON

**Input:** *Input data overview*

identifier is the id of an ingested item

**Output:** *Response data overview*

An empty Json entity (“{ }”) if identifier does not exist.

**Codes:** *HTTP status codes*

- 405 no identifier
- 200 item with given id exists
- 404 item does not exist
- 500 server error

### 2.1.2 State change

Posting that user selected a new item (state change!). Security information is provided in the HTTP Headers.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/access/{identifier}		TEXT-XML

**Input:** *Input data overview*

identifier is the id of an ingested item

**Output:** *Response data overview*

the identifier if successful (as plain text)

Example: /access/6HLLSQDFEPURYZ4VUROUXG5UDY46AIM4/  
6HLLSQDFEPURYZ4VUROUXG5UDY46AIM4

**Codes:** *HTTP status codes*

- 200 item with given id exists
- 404 item does not exist
- 500 server error

### 2.1.3 Listing All Components

List all available components for an item. Security information is provided in the HTTP Headers.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/access/{id}/components		APPLICATION-JSON

**Codes:** *HTTP status codes*

- 200 item with the given id exists
- 500 server error

**Input:** *Input data overview*

identifier is the id of an ingested item (or any other non empty string)

**Output:** *Response data overview*

Returns the following list of components (independent from id).

```
{
  "components":
  ["item-binaries", "ingest-id", "source", "cortex-type", "place", "report",
  "providerinfo", "title", "item-id", "journeys", "model", "ingest-date", "ipr",
  "preview", "indexing-profile", "view", "media-types", "epoch"]
}
```

Listing 2.1: Components

## 2.1.4 Listing Single Components

Get a component of an AIP. Components can be preview, view, metadata, etc. If the component or the AIP is not found a `WebApplicationException` resp. a 404 will be thrown. Security information is provided in the HTTP Headers.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/access/{id}/components/{component}		APPLICATION-JSON

**Codes:** *HTTP status codes*

200 item component with the given id exists  
 404 item or component does not exist  
 500 server error

**Input:** *Input data overview*

identifier is the id of an ingested item

component is one of the strings defined in components

**Output:** *Response data overview*

Returns an empty Json entity (“{}”). if item does not exist or if there are no components.

Returns a json-file containing xml-formated text or plain text.

Example /access/{id}/components/place

```
<place>
  <name>Ulm</name>
  <latitude >48.398312</latitude >
  <longitude >9.9910464</longitude >
```

</place>

### 2.1.5 Listing Subcomponents

Returns a subcomponent of an item. Security information is provided in the HTTP Headers.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/access/{id}/components/{componentName}/{subComponent: .+}		APPLICATION-JSON

**Codes:** *HTTP status codes*

- 200 item component with the given id exists
- 404 item or component does not exist
- 500 server error

**Output:** *Response data overview*

Returns an empty list if item does not exist or if there are no components.

## 2.2 User Profiles

The user profile interface handles the users of the system. A user profile does not handle access rights nor passwords but lets the user open up additional functions of the system. At the moment "external" services can be registered here.

### 2.2.1 Application: Register

TODO: Describe this mechanism.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/registerapplication		TEXT-PLAIN

**Codes:** *HTTP status codes*

- 200 all ok
- 500 server error

**Output:** *Response data overview*

The application id assigned to the given application.

## 2.2.2 Application: Register

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/unregisterapplication/{appid}		TEXT-PLAIN

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

The application id assigned to the unregistered application.

## 2.2.3 Topics: Add

Add one or more topics to a registered application. TODO: Describe this mechanism properly.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/topics/{appid}/{topic}		TEXT-PLAIN

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

All topics assigned to the application.

## 2.2.4 Topics: Get

Gets topics assigned to the application.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/topics/{appid}		TEXT-PLAIN

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

All topics assigned to the application.

### 2.2.5 Topics: Remove

Removes topics assigned to the application. Topics may be comma-seperated. TODO: Describe this mechanism properly.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
DELETE	/userprofile/topics/{appid}/{topic}		TEXT-PLAIN

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

The application id assigned to the application.

### 2.2.6 Journeys: Get head

Retrieves the last added update.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/version		APPLICATION-JSON

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

The last added update as Json.

### 2.2.7 Journeys: Deactivate lock

Deactivates the lock. If deactivated, incoming Updates are added to the history queue.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/history/turnon		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.8 Journeys: Activate lock

Activates the lock. If activated, incoming Updates are not added to the history queue, but affect current state.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/history/turnoff		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

## 2.2.9 Journeys: Is lock activated

Retrieves the state of the lock.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/history/isturnedon		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

One of two Json Strings, which may be “{writehistory:false}” or “{writehistory:true}”.

## 2.2.10 Journeys: Add Update

Adds an Update. Depending on the history lock, it is added to the history queue. It is always used to adjust the current state. parentId identifies the parent under which the added node is placed in the Update tree.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/add/{component}/{content}/{parentId}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

## 2.2.11 Journeys: Delete Update

Adds an Update. Depending on the history lock, it is added to the history queue. It is always used to adjust the current state. parentId identifies the parent under which the added node is placed in the Update tree.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/journeys/delete/{updateId}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

## 2.2.12 Journeys: Add Named Journey

Adds a namedJourney to the user’s JourneyData. It consists of a user-unique namedJourney-Name, the time-stamp of the LAST Update and its number of ancestor Updates.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/namedjourneys/add/{namedJourneyName}/{lastUpdateId}/{numberOfPredecessors}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

## 2.2.13 Journeys: Rename Named Journey

This method renames the old namedJourney to the new name.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/namedjourneys/rename/{oldNamedJourneyName}/{newNamedJourneyName}		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.14 Journeys: Delete Named Journey

Deletes the given namedJourney from the user’s JourneyData.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/namedjourneys/delete/{namedJourneyName}		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.15 Journeys: Get Named Journey

Retrieves the Update nodes of the given namedJourney. The result is sorted from LAST Update upwards.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/namedjourneys/get/{namedJourneyName}		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

Returns concatenated Json of the Named Journey’s Updates. Delimiter is ‘+’.

## 2.2.16 Journeys: Get Data

Delivers the whole JourneyData as Json! This includes the Update history, named journeys begin nodes and predecessors, the current state map and history lock.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/namedjourneys/get/{namedJourneyName}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns the whole data as Json.

## 2.2.17 Journeys: Erase Data

Erases the user's journey data.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/erasedata		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

## 2.2.18 Journeys: Get Update for timestamp

Searches the journey history for an Update with the given timestamp.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/get/{timestamp}		application/json

**Codes:** *HTTP status codes*

- 200 all ok
- 500 server error

**Output:** *Response data overview*

The found Update as Json. An empty String if there was no Update matching the timestamp.

### 2.2.19 Journeys: Get Updates between timestamp

Searches the journey history for all Updates between the given timestamps. Updates hitting exactly startTime or endTime are included.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/get/startTime/endTime		application/json

**Codes:** *HTTP status codes*

- 200 all ok
- 500 server error

**Output:** *Response data overview*

The found Updates as Json. Delimiter is the character '+'. Returns an empty String if there was no Update matching the timestamp.

### 2.2.20 Journeys: Store Journey Data

Triggers a persistent write of Journey Data. Target and method may depend on configuration.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/journeys/store		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.21 Journeys: Get timestamps

Retrieves timestamps of all Updates in the journey history.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/gettimestamps		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns concatenated timestamps of the found Updates. Delimiter is “+”.

### 2.2.22 Journeys: Get current state for a topic

Fetches the Update linked to the given topic from current state Map.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/getcurrentstate/{topic}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns an Update as Json. An empty String if there is none for this topic.

### 2.2.23 Journeys: Get all current states

Gives all Updates in the current state Map.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/getcurrentstates		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

Returns all Updates as Json. Delimiter is "+". Error 404 if the map is empty!

### 2.2.24 Journeys: Get all current state timestamps

Gives the timestamps of all Updates in the current state Map.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/currentversions		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

Returns all timestamps. Delimiter is "+". Error 404 if the map is empty!

### 2.2.25 Journeys: Get all breadcrumbs

Gets all Updates in the user's history.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/journeys/getbreadcrumbs		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns concatenated Json of Updates. Delimiter is '+'.

### 2.2.26 Journeys: Get latest breadcrumbs

Gets the latest Updates in the user's history. Latest is currently defined as 10.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/history/breadcrumb		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns concatenated Json of Updates. Delimiter is '+'.

### 2.2.27 Collections: Get collection names

Retrieves all names of the user's collections.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/collections		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns the concatenated names. Delimiter is '+'. Returns an empty String if the user has no collections.

### 2.2.28 Collections: Delete collection

Deletes the collection with the given name.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
DELETE	/userprofile/collections/{collectionId}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.29 Collections: Add item to collection

Adds the given item to the collection. If it does not exist yet, the collection is created automatically.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
PUT	/userprofile/collections/{collectionName}/{item}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.30 Collections: Delete from collection

Deletes the given item from the collection.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
DELETE	/userprofile/collections/{collectionName}/{item}		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.31 Collections: Has item in collection

Checks if the given item is in the collection.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/collections/{collectionId}/has/{item}		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

A plain text String which can be “true” or “false”.

### 2.2.32 Collections: Get collection

Gets all items in the collection.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/collections/{collectionId}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns the concatenated items. Delimiter is '+'. Returns an empty String if the collection has no items.

### 2.2.33 Collections: Get collection size

Gets the number of items in the collection.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/collections/{collectionId}/size		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns the concatenated items. Delimiter is '+'. Returns an empty String if the collection has no items.

### 2.2.34 Collections: Get collection's items from start position on

Retrieves items from the collection, beginning from startrow position on for count items.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/collections/{collectionId}/{startrow}/{count}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns the concatenated items. Delimiter is '+'. Returns an empty String if the collection has no items.

### 2.2.35 Collections: Store collection data

Triggers a persistent write of collection data. Target and method may depend on cortex configuration.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/collections/store		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.36 Configurations: Add item to configuration

Adds the given item to the configuration. If it does not exist yet, the collection is created automatically.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/configurations/add/{item}		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.37 Configurations: Delete item from configuration

Deletes the given item from the configuration.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/configurations/delete/{item}		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.38 Configurations: Has item in configuration

Checks if the given item is in the configuration.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/configurations/has/{item}		application/json

**Codes:** *HTTP status codes*

200 all ok

500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.39 Configurations: Get configuration

Retrieves all user’s configuration items.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/configurations/get		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

All configuration items concatenated as Strings. Delimiter is “+”.

### 2.2.40 Configurations: Store configuration

Triggers a persistent write of configuration data. Target and method may depend on cortex configuration.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/configurations/store		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.41 Services: Add to Service

Adds the given item to the user’s service data.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/services/add/{item}		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.42 Services: Delete from Service

Deletes the given item from the user's service data.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/services/delete/item		application/json

**Codes:** *HTTP status codes*

- 200 all ok
- 500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

### 2.2.43 Services: Has item in Service

Checks if the given item is in the services.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/services/has/{item}		application/json

**Codes:** *HTTP status codes*

- 200 all ok
- 500 server error

**Output:** *Response data overview*

A plain text String which can be “true” or “false”.

### 2.2.44 Services: Get Service

Retrieves all user's configuration items.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/userprofile/services/get		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

Returns the concatenated items. Delimiter is '+'. Returns an empty String if the service has no items.

## 2.2.45 Services: Store Service

Triggers a persistent write of configuration data. Target and method may depend on cortex configuration.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/userprofile/services/store		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

An empty Json entity (“{}”).

## 2.3 Search and Facets

### 2.3.1 Normal Search

Searches the index and returns the results as json. If the **facet** option is used, the query is filtered by this facet. Returns only those results that match the facet.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/search?query={query}[&offset={offset}] [&rows={rows}][&minDocs={minDocs}] &facet={facetField}*		application/json

**Codes:** *HTTP status codes*

200 is always returned

500 server error

**query:** Contains the query. Should be URL escaped

**offset:** Offset parameter for pagewise navigation through search results.

**rows:** Rows parameter for pagewise navigation through search results.

**minDocs:** The minimum count of documents the facets have to contain.

**facetField:** The name of the facet.

**facetValue:** The value of the facet.

**Output:** *Response data overview*

Returns an empty list if no documents can be found.

```
{
  "results":
  [
    {
      "id":"101",
      "preview":" Preview"
    },
    {
      "id":"102",
      "preview":""
    }
  ],
  "facets":[
    {
      "field":" title",
      "facetValues":
      [
        { "value":" Alpina", "count":22},
        { "value":" Tabula", "count":1},
      ],
      "numberOfFacets":2
    },
    {
      "field":" date",
      "facetValues":
      [
        { "value":" 1970", "count":19},
        { "value":" 1853", "count":2},
      ],
    }
  ]
}
```

```

    "numberOfFacets":2
  }
]
"numberOfResults":2
}

```

Listing 2.2: Search result

**results:** JSON Array containing the documents found. Empty if no documents can be found. Otherwise it contains the IDs and a preview of the documents.

**id:** ID of a found document. Can be used to access other components of this document.

**preview:** Preview of a document delivered for speed reasons.

**numberOfResults:** Count of the documents found.

**field:** The name of the facet

**facetValues:** contains the representations

**value:** The name of the facet representation

**count:** Count of the facet representations

**numberOfFacets:** Count of the facets.

### Example:

```

/search?query=tabula%20rasa&offset=100&rows=200&minDocs=2&facet=ort&ort=St.%20Augustin&ort=
Berlin

```

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
POST	/search/		APPLICATION-JSON

Very similar to the GET-search.

### 2.3.2 Cluster Search

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/search/cluster		APPLICATION-JSON

## 2.4 Ingest

Directly ingests outboxes on the filesystem into Cortex bypassing the conventional way via ASC.

### 2.4.1 Ingest A SIP

Store an AIP into the Core-Library.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
PUT	/ingest/	TEXT-XML	

**Codes:** *HTTP status codes*

- 201 on successful ingest+
- 303 if the ingest process failed for some reason
- 415 Unsupported Media Type
- 500 server error

**Input:** *Input data overview*

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/ingest/directingeststart?outbox={outbox}		TEXT-PLAIN

Starts an ingest via GET request.

**outbox:** Path to the outbox

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/ingest/directingeststop		TEXT-PLAIN

Stops a running ingest identified by either outbox directory or thread id

## 2.5 Reporting Interface

The reporting interface provides a view into the ingest process. This serial process is splitted into single process fragments. Every fragment examines an aspect of the ingest (i.e. recognizing

facets). On the process fragments layer there exists an event-check, that prevents an uncontrolled cancellation in case of an error. At this point the reporting mechanism is attached and takes over the status in a report. After completion of all process fragments or after an error has occurred the report is persisted and can be queried by a service.

So far there exist two types of reports single reports for each item

### 2.5.1 Listing A Single Report

The single report informs about a single ingested item. It contains besides the global status of the ingest a more detailed view on secondary parameters, like for example the duration of the ingest.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/reports/{ingest-id}/{item-id}		text/xml

By creating a report /reports/{ingest-id}/{item-id} are deleted. These reports are still available via /reports/item/{item-id}

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/reports/item/{item-id}		text/xml

**Codes:** *HTTP status codes*

- 200 item report with the given id exists
- 404 item or report does not exist
- 500 server error

**Output:** *Response data overview*

Returns an empty list if item does not exist or there are no sub-ids.

```
<cortex-report>
  <ingest-event-id>1297450809</ingest-event-id>
  <item-id>2012431732</item-id>
  <ingest-time unit="ms">1891</ingest-time>
  <provider>
    <provider-name>http://gdz.sub.uni-goettingen.de/oai2</provider-name>
    <provider-item-id>
      http://resolver.sub.uni-goettingen.de/purl?PPN38180691X_Tafeln
```

```
    </provider-item-id>
  </provider>
  <ingest-status>
    <status>success</status>
    <error-count>0</error-count>
    <success-count>4</success-count>
  </ingest-status>
  <success-list>
    <success index="1">
      <message>
        CortexModelDomWorker de.fhg.iais.cortex.services.normalizing.postbinding.
        started
      </message>
    </success>
    <success index="2">
      <message>
        CortexModelDomWorker de.fhg.iais.cortex.services.normalizing.postbinding.
        Started
      </message>
    </success>
    <success index="3">
      <message>
        CortexModelDomWorker
        de.fhg.iais.cortex.services.normalizing.postbinding.DownloadableLinkRepla
        started
      </message>
    </success>
    <success index="4">
      <message>Object with new item id 2012431732 successfully ingested</message>
    </success>
  </success-list>
  <error-list />
</cortex-report>
```

Listing 2.3: Single report output

## 2.5.2 Listing All Reports

In addition to the single view of the items an aggregated ingest event report can be queried. For this kind of report all single reports are merged. **Currently only the global status of the items is returned.**

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/reports/{ingest-id}[?uri=true]		text

**uri** *Optional* if not given or false, only a list of ids is returned. Otherwise the list is returned with complete URIs. **Parameters are not used yet**

**Codes:** *HTTP status codes*

200 item report with the given id exists  
500 server error

**Output:** *Response data overview*

Returns

if ingest does not exist or there are no sub-ids.

All single reports accessible via /reports/{ingest-id}/{item-id} are deleted. These reports are still available via /reports/item/{item-id}

```
# Ingest Report
## Ingest Event: 1299577852

- **Item:6HLLSQDFEPURYZ4VUROUXG5UDY46AIM4**
  - success
- **Item:EG3ZP33ZRNTJTCSLFO4ORIWSNDSEV34H**
  - success
- **Item:EXUZMRCL645CCZDXUGCB5AAZVZWSVF3E**
  - success
- **Item:HXPYZKJWKJ4F3FT3GOYPQNZJJMD3KYP**
  - success
- **Item:I74K7JV3QV5ODVSYNNURKWMQFWAHXQD2**
  - success
- **Item:LFVG4VDNQ4FKTR44TOJOMHJSAJ3NTGCU**
  - success
- **Item:RFQHOKACSATYX6U6Y52UA4S2X3MKPSOB**
  - success
- **Item:RRK6NSAWDXKZ4L3AQ77ADRXVS5ANPGEV**
  - success
- **Item:UBLOGZ2TVNDIMDIZ6LAVMFEW654VKKFO**
  - success
- **Item:UYGM4JGPXXX6UTN4WXH7LTJW2KSYMT27**
  - success
- **Item:V4MPFV7LWGGDXIDEYXPEA3MM2DA3DGAM**
```

– success

Listing 2.4: Ingest event report

## 2.6 Hopping

### 2.6.1 Retrieving first order relations

Retrieves all first order relations of a given type for a given entity.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/hopping/{identifier}		Application/JSON

**Codes:** *HTTP status codes*

200 ok

500 server error

**Output:** *Response data overview*

Returns an empty list if item does not exist.

```
{ "any" : [
  { "resource" : " http://www.ddb.de/557822501" },
  { "resource" : " http://www.ddb.de/557822501" },
  { "resource" : " http://www.ddb.de/2620479854" },
  { "resource" : " http://www.ddb.de/3291657937" }
]}
```

Listing 2.5: Single report output

### 2.6.2 Retrieving preconfigured first order relations

Retrieves all first order relations of a given type for a given entity and a configuration value.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/hopping/{identifier}/{configKeyname}		Application/JSON

**Input:** *Input data overview*

**configKeyname:** one of the preconfigured targets in Cortex/appctx/hopping/hopping\_config.xml. Defaults to "any".

**Codes:** *HTTP status codes*

200 ok  
500 server error

**Output:** *Response data overview*

A JSON object containing only the identifiers of the nodes that are tied to the given id through the given relation type. May be empty but never is null.

```
{ "title_hop": [ { "resource": "http://www.ddb.de/557822501" } ] }
```

Listing 2.6: Configured report output

### 2.6.3 Fuzzy Retrieve all destinations of a given type for a given entity using the search index.

Retrieve all destinations of a given type for a given entity using the search index. **TODO: verify**

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/hopping/fuzzy/{relation}/{literal}		Application/JSON

**Codes:** *HTTP status codes*

200 ok  
500 server error

**Output:** *Response data overview*

A JSON object containing only the identifiers of the nodes that are tied to the given id through the given relation type. May be empty but never is null.

```
{ "title_hop": [ { "resource": "http://www.ddb.de/557822501" } ] }
```

Listing 2.7: Configured report output



## 3 Augmented SIP Creator (ASC)

### 3.1 Introduction

The purpose of the Augmented SIP Creator (ASC) is to load object records from different sources to transform them into Submission Information Packages (SIPs). These SIPs are transferred and ingested into the Cortex. From the birds eye perspective the ASC can be seen as an **extract, transform, load-tool** (ETL-tool) <sup>1</sup>.

Our ASC is a commandline implementation of an ASC. The ASC includes a OAI-PMH harvesting component, a preformatter, the necessary transformation component as well as a component sending the data to the Cortex.

### 3.2 Building the ASC

To build the ASC *maven* is used. Calling `mvn clean package assembly:assembly` on the parent project ASC compiles the build and stores it under `ASC/target/asc-0.0.1-SNAPSHOT-asc`. The contents of this directory has to be copied completely to a harvesting machine and can then be started from there. Bundles like *myCortex* and *cortexAssembly* include the ASC.

### 3.3 Directory structure for storing metadata

#### 3.3.1 Directory structure defaults

The directories of the ASC are structured as following:

**INBOX** (contains downloaded files in original state):

`inbox/metadatenformat/ingest_event`

**WORKSPACE** (contains splitted and preprocessed files):

`workspace/metadatenformat/ingest_event`

**WORKSPACE.TMP** (contains the results of the several XSL transformations (XSLTs) except for the last):

---

<sup>1</sup>P. Vassiliadis, A. Simitsis. Extraction, Transformation, and Loading. Encyclopedia of Database Systems. Editors-in-chief: Ling Liu and M. Tamer Ozsu, Springer, 2009.

```
workspace.tmp/metadatenformat/xslt-dateiname/ingest-event
```

**OUTBOX** (contains the results of the last XSLT transformation, which represents the SIP for the ingest):

```
outbox/metadatenformat/ingest_event
```

In the particular ingest-event directories you can find subfolders containing the respective sets. These set-folders can also be created by the OAI-PMH-harvester. Every record retrieved from the OAI-PMH repository is saved in one file. During the process every record is saved in one file.

### 3.4 Controlling ASC

The ASC-headless can be started via commandline by the following command:

```
java -jar asc.jar provider.properties-File
```

Omitting the properties-file parameter ensures automatic use of the `provider.properties` file. The global configuration is deposited in the file `asc.properties`. Both files are mixed together to gather the configuration of the ASC. This is done by first loading the `asc.properties` contents and then loading the `provider.properties` contents. Values which are already loaded from the `asc.properties` and are repeated in the `provider.properties` file are overwritten by the values in the `provider.properties` file. If you want to adapt your global configuration within the `provider.properties` files please feel free to copy all global configurations within the `provider.properties` files leaving a blank `asc.properties` file.

The parameter `properties-file` supports wildcards, so that multiple properties-files can be processed consecutively.

Within the `provider.properties/asc.properties-file`, you can find the configuration for operating the ASC:

```
host:http://localhost:9998
oai_source:http://gdz.sub.uni-goettingen.de/oai2/
oai_set:
maxfilesEach:10
inbox:../__repository/asc/inbox
workspace:../__repository/asc/workspace
outbox:../__repository/asc/outbox
ingest_event:1295424833
do_download:true
do_transform:true
do_send:false
harvestingstrategy:multirecord
```

```
cleanup : false
oaipmhproxyhost :
oaipmhproxyport :
harvesting_format : oai_dc
source_folder : oai_dc
md_format : oai_dc , b , , 1
```

Listing 3.1: Properties Configuration

### The parameters in detail:

Property	Description of the property
host	URL of the ingest endpoint of the DDB
oai_source	URL of the OAI-PMH endpoint
oai_set	Sets, that are supposed to be harvested from the OAI-PMH interface. Leaving this parameter empty results in harvesting all sets.
md_format	Handlers/Transformers configuration selection. The desired configuration is separated by commata. The values seperated by commata in order: metadata format, provider- 'Sparte', provider name, collection name, version number. E.g. <i>oai_dc, , , 2</i> means <i>oai_dc-metadataformat</i> and configuration version 2 which is located in the file <i>oai_dc/1.xml</i> <i>OR</i> <i>ese, b, staabi – berlin, , 1</i> means <i>ese-metadataformat</i> , Sparte 'b' for 'Bibliothek', 'staabi-berlin' for provider name, and 1 for version which is located in the file <i>ese/b/staabi – berlin/1.xml</i> .
maxfilesEach	Number of the metadatafiles, that are to be processed. For testing purposes.
inbox	The INBOX directory to store or expect the harvested files.
workspace	The WORKSPACE directory for storing the preprocessed files.
outbox	The OUTBOX directory to store the SIPs to be sent to the Ingest.
ingest_event	The ingest second. This parameter has to comply the corresponding directory in the filesystem. Defaults to the current time. Unsetting this parameter only makes sense if the data is sent through the whole process chain, in other words Harvest, Preprocessing, Transformation, Ingest.
do_download	true or false. Declares whether to harvesting
do_transform	true or false. Declares whether to transform
do_send	true or false. Declares whether to ingest
harvestingstrategy	<b>chunk</b> or <b>completeset</b> or <b>multirecord</b> .

	<p><b>chunk</b> loads a partial list of identifiers and then gets the records of the specific identifiers, before the next partial list is processed. <b>completeset</b> loads the whole list first and caches it in <b>identifiers.x.txt</b> before the records are downloaded one at a time. Resuming is possible. <b>multirecord</b> harvests the records directly.</p>
cleanup	Automatic cleanup of the transformation directories
oaipmhproxyhost	Proxy-Configuration for the OAI-PMH
/ oaipmhproxyport	interface
harvesting_format	This format is requested via OAI-PMH interface
source_folder	The local destination folder for the data.

The configuration of the handlers / transformers selected with `md\_format` is loaded from the filesystem structure within `conf\appctx\asc\sipmakers` :

```

+— ead
|   +— 17072011
|   |   +— 1.xml
|   +— 1.xml
+— ese
|   +— 1.xml
|   +— b
|       +— 1.xml
|       +— staabi-berlin
|           +— 1.xml
|           +— 2.xml
+— lido
|   +— 17072011
|   |   +— 1.xml
|   +— 1.xml
+— marc
|   +— 1.xml
+— mets
|   +— 1.xml
+— museumdat
+— oai_dc
|   +— 17072011
|   |   +— 1.xml
|   +— 1.xml
|   +— b
|   |   +— 1.xml
|   |   +— wikipediademo

```

```

|   |   +--- 1.xml
|   +--- new
|   +--- 1.xml
+--- oai_dif
     +--- 1.xml

```

Listing 3.2: Sipmaker Configurations

Each configuration is written down as a Spring-configuration like the following for MARC-XML. Within the configuration you see that a splitter of the original files named `marcSplitter` is used. After that the preprocessor `xmlfieldstripper` is running. Within the `transformers` property you see all running XSLT-scripts separated by commas. These scripts can be found in the `transformers` directory:

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd">

  <bean id="sipmaker" class="de.fhg.iais.asc.sipmaker.SipMakerImpl">
    <property name="splitter" ref="marcSplitter" />
    <property name="xmlProcessors">
      <list>
        <ref bean="xmlfieldstripper" />
      </list>
    </property>
    <property name="transformers"
      value="marc21_create , marc21_identify , MARC21slim2MODS3-3,
            mods_crm , mods_views" />
  </bean>
</beans>

```

Listing 3.3: One Sipmaker Configuration

### 3.5 myCortex

The ASC commandline tool comes with a SWING-GUI component which allows the configuration of the property files graphically. The GUI component writes and reads the original property

files of the ASC. In addition, it starts a local Cortex server which accepts the ingests and therefor provides a preview of the Cortex.

### 3.5.1 Start-Up

The myCortex application is implemented as a Java application. This means that you must have the Java Runtime Environment installed on your system. All Java Runtime Environment versions including and above 1.6 are supported.

Additionally you need a network connection on your computer to harvest data from your OAI-PMH-server and transfer it to the Cortex.

Some operating systems may ask you if you allow using connections to local ports. In the background it starts up a server which will accept the ingests, makes your metadata searchable and displays your objects as if they were transfered to the Cortex. Please configure your operating system the way that the ports are unblocked and usable but aren't accessible by intruders from the internet.

### 3.5.2 Working with myCortex

The myCortex user interface consists of two columns. You can move your mouse over the fields to get a short help message.

On the left side the meta information about the metadata you want to ingest has to be filled in into the provided fields. Especially you have to define which steps of the process should take place:

1. *Harvesting:* You can specify an OAI-PMH-server. If you do not want to harvest your data but have it already transfered as local files you can skip harvesting by placing your data in INBOX/Metadataformat where Metadataformat is the metadataformat of the files.
2. *Transformation:* The metadata is transformed with XSLT-stylesheets. You can choose if these stylesheets are downloaded from the central server or if the local stylesheets should be used. You will find the local stylesheets in the directory transformers.
3. *Sending:* You can send the transformed metadata to the local server to preview the results or your can send it to the central server of the DDB.

On the right side you have control over the process to start it and get error reports. On the top you see the main start/stop button. After you entered all data on the left correctly please press this button to start the ingest process. Under the start/stop button you see a progress bar and a window reporting errors.

Three additional buttons are available below the error reports:

1. *Show ingest report:* To show a report of the ingest.

2. *Show preview*: To show the local preview of your ingested data. You will see an empty Cortex server if you ingest anything.
3. *Discard preview*: This deletes the local server. You can start from the scratch.



## 4 Monitoring

This chapter handles the monitoring of the running server instances / services in the project "Deutsche Digitale Bibliothek" (DDB).

### 4.1 Overview

In order to measure the load factor and the data throughput of all relevant services, dedicated components are to be written to perform these measurements and to display the results to a user in a webapplication in an interpreted form.

Components to be developed:

1. AdminServer
2. ICortexServiceMonitor + Implementation(s)

### 4.2 AdminServer

Extending the `AbstractCortexServer` (`de.fhg.iais.cortex.rest.server.AbstractCortexServer`), the `AdminServer` is a Jetty-based webserver, which, besides provisioning the `UserInterfaces` is in charge for communication with the service-monitors and processing the delivered measured data in an appropriate manner.

You can find the sources of the `AdminServer` in the DDB subproject "ServerAdminConsole".

When running Cortex as a single-machine installation you can access the AdminServer's web console via `http://localhost:8070/admin/`. The displayed page offers links to detail pages for monitored services and servers. All pages are updated automatically every 10 seconds. The following subsections describe all available pages in detail.

#### 4.2.1 Servers overview

This is currently the homepage of der Cortex Admin Console. Its layout is divided in three parts. On the top there is - besides the name of the currently displayed page - a list of links which lead to other admin pages. Below that list there is a table showing all members of the monitored cluster. In this context a cluster is a group of one or more Java Virtual Machines accessible to each via Hazelcast. For each cluster member the host name and the port used by Hazelcast is

shown. On the page's bottom there is a second table showing all different servers running in the monitored clusters. Each server is identified by its server type in the table's first column. The second column shows the host name the server is running on. The third and last column provides a link to a page showing details of the server.

### 4.2.2 Server details

For each server a page is provided that shows detailed informations of the running instance. Below the link list a single rowed table gives a description of the selected server with the server's type, its id, the host's name and its IP port (if available). Next there is a chart showing the memory usage in percent of the JVM's available memory, the available disk space on the server in percent and the current CPU usage, also in percent (Note: As of now the CPU meter is kind of useless since the JVM versions prior to 1.7 can only inspect CPU times in relation to processing times assigned to the JVM by the operating system; thus a typical value of 80). Besides the chart there is a table showing the server's uptime, the number of running threads, the amount of used / available memory and the amount of free / total disk space. The bottom of the page consists of two tables showing informations about the operating system (OS Name, version, architecture, number of processors and, depending on the OS, the system load) and the JVM the server is running on (server name, JVM name, vendor, version, specification name, specification vendor and specification version). When running Cortex as a single machine installation all server detail pages will show nearly the same content since all server instances are running inside the same JVM.

### 4.2.3 Services

This page can be accessed by following the link to "Services" on the top of each page. The page consists of a table showing a list of all monitored service by their type, the number of running instances and the corresponding service id. A click on one of these ids opens a new page with details for the chosen service. By clicking on a service type a summary page for this type opens.

### 4.2.4 Service summary

The service summary page shows summed up informations of all instances of the given service type. The kind of infos shown on this page depends on the actual service type. On the page's top there's a table which typically provides number for

**Objects count** the number of objects processed by all service instances of the given type since start of the AdminServer

**Object size** the sum of the sizes of all objects processed by the service's instances. The actual meaning of 'size' depends on the service type. For example the size of an Ingest object is the number of character in the payload of the SIP

---

**CPU time** the amount of processing time that the operating system(s) assigned to the services

**User time** the actual amount of processing time used by the services

**System time** the amount of wall time

**Throughput** the average of objects processed per second by the service

On the ingest service summary page there is another table showing technical errors that may have occurred while ingesting. For each error a timestamp (when did the error occur?), the id of the ingest event and an abbreviated error description is shown. A click on the error's short text shows its complete stack trace.

#### 4.2.5 Service details

Detailed information for each instance of a monitored service can be viewed on a separate page in the Cortex Admin Console. The appearance of those pages is similar to each other, independent of the specific service type. On top of the page some general information about the chosen service is available (the service type, its id, the host name it's running on and the port used by Hazelcast to connect the service instance to the AdminServer). Below is a chart showing the service's current activity and a table with statistical data concerning the service. At the bottom there are one or two tables with information about the most recent operations executed by the service. The meaning of the numerical values depends on the service type, so following is a description for each type in turn.

##### Access

STATS table:

**# of accesses** the number of access events for the last 300 seconds

**Avg. t/access (ms)** the average amount of time it took to process each access, measured in milliseconds

**Avg. size of access** the average size of accessed objects measured in bytes

LAST 20 DATA table:

**Submitted** when has the access occurred

**Time (ms)** the amount of time the service needed to process the access

**No. of results** the count of accessed objects in this event

## Ingest

STATS table:

**# of ingests** the number of ingest events for the last 300 seconds

**Avg. t/ingest (ms)** the average amount of time it took to process each ingest, measured in milliseconds

**Avg. size of sip** the average size of the ingested sips, measured in number of characters of the payload

LAST 20 DATA table:

**Submitted** when has the ingest occurred

**Time (ms)** the amount of time the service needed to process the ingest

**Size** the sips size

**Unit** the measure unit of the Size column

TIMING FOR SUBEVENTS table:

**Subeventtype** every ingest is divided into several steps. The subevent type reflects those steps, giving them a name

**Gliding avg. (ms)** the gliding average of the amount of time each step took to complete taken over the last 100 events

**Time (ms)** the amount of time each step took

**Tendency** either a green or a red sign. Red meaning the most recent step took longer to complete than the gliding average, green meaning the contrary.

## Messaging

OVERALL table:

**Processed Message Count** the total number of messages that a had been processed

**Message size** the average size of the ingested sips, measured in number of characters of the payload of the message

**CPU time** the amount of processing time that the operating system(s) assigned to the services

**User time** the actual amount of processing timew used by the services

**System time** the amount of wall time

**Throughput (Messages/Second)** the average of messages processed per second by the service

**SETTINGS table:** This table show some input fields which can be changed by an administrator. After pushing the 'Submit changes' button the new values are applied to the live production environment.

**throttling - pause between ingest** (milliseconds) - a throttling for each thread which polls messages from the message queue and pushes those into the ingest method. This value should be 0 (zero) in production environment.

**number of ingest processing threads** the number of threads which are polling on the message queue and pushing them to the ingest method. This value shall be equal to the number of processor cores in the production environment. It is possible to set the value to -1 to enable an automatic adoption to the actual number of processing cores.

**message replication enabled** (flag) - if this is set, then all messages are stored to a replication queue where another storage process reads this messages and stores them into a storage path for replication (see below).

**storage path for replication** This is the path where messages from the replication queue is stored. There can be several replication queues, but then several storage threads must be started, one for each of a storage queue.

**ACTIVE MQ QUEUE MONITOR table:** this is a monitor for messages in the ActiveMQ messaging server which have not been processed so far. There is a deletion button in each row which can be used to delete the corresponding message queue completely.

**ingestsubmit** message pending to be processed in an ingest process

**replication** messages pending to be processed in an storage process

## Search

**STATS table:**

**# of searches** the number of search events for the last 300 seconds

**Avg. t/result (ms)** the average amount of time it took to process each search, measured in milliseconds

**Avg. # of hits** the average size of the search's result

**LAST 20 DATA table:**

**Submitted** when has the access occurred

**Time (ms)** the amount of time the service needed to process the search

**No. of results** the number of objects found in this search

## SPARQL

STATS table:

**# of sparql acc.** the number of sparql access events for the last 300 seconds

**Avg. t/acc. (ms)** the average amount of time it took to process each sparql access, measured in milliseconds

**Avg. size of data** the average size of the accessed data, measured in bytes

TIMING OF SPARQL UPDATES table:

**Eventtype** the specific type of the sparql update request

**Gliding avg. (ms)** the gliding average of the amount of time the service took to complete taken over the last 100 events

**Time (ms)** the amount of time the update took

**Tendency** either a green or a red sign. Red meaning the most recent update took longer to complete than the gliding average, green meaning the contrary.

TIMING OF SPARQL EXECUTES table:

**Eventtype** the specific type of the sparql execute request

**Gliding avg. (ms)** the gliding average of the amount of time the service took to complete taken over the last 100 events

**Time (ms)** the amount of time the execute took

**Tendency** either a green or a red sign. Red meaning the most recent update took longer to complete than the gliding average, green meaning the contrary.

## 4.2.6 Thread dump

## 4.2.7 Profiling

## 4.2.8 Solr

## 4.2.9 Settings

The AdminServer can be configured to send Emails if some special conditions are met. An email address and a threshold can be specified for every server type. If the memory usage exceeds the one shown under 'Threshold' a mail is send to the provided email address. The same can be configured for the different service types. If a service's throughput gets higher than the one configured a mail is send to the provided address.

### 4.3 ICortexServiceMonitor

This interface is part of the package `de.fhg.iais.cortex.services.monitor` in the subproject `CW_Interface` and exposes a number of methods of which implementations deliver functionalities for investigating of the diverse measured data. To be measured are temporary values (three different kinds of time consumption per service), data volume (service dependent) and VM-specific values like memory consumption and available memory. Planned methods for a first draft are:

**public MonitoringData start(MonitoringData md)** starts the internal stopwatches, measures the current memory expendage and returns the modified monitoring data. If the provided `md` was `null`, a new object will be created.

**public void stop(MonitoringData md)** stops the current measurement and submits the results to the AdminServer.

**public void pause(MonitoringData md)** pauses the internal stopwatches.

**public void resume(MonitoringData md)** continues the internal stopwatches.

**public MonitoringData measure(MonitoringData md, Object data)** calculates data-specific sizes like payload volume, etc. of `data` and adds the measurement to `md`.

**public MonitoringData addMeasure(MonitoringData md, MeasureType mt, int size)** adds a new measurement to `md` with measure type `tp` and measured size `size`.

**public MonitoringData newMeasurement()** creates a new transport object for measurements.

**public ServiceType getServiceType()** returns the type of service this monitor is watching.

**public ServerType getServerType()** return the type of server this monitor is watching.

In a first step the `CortexDomEvaluationChainMonitor` is implemented, providing measurement points for the `IngestService`, replacing the already existing `IngestMonitor` (`de.fhg.iais.cortex.service`). The data volume to be measured consists in this case of the user data of the `AIPDom` or parts of it. In addition the `IngestServiceMonitor` can be configured via spring with an arbitrary count of XPath-expressions, that specify the tree nodes for every measurement, of which textual content should involve in the volume measurement. Initially there will be two measurement points in the `IngestService`: one right after creation of the `AIPDom`, and one after it's successful enrichment.

## 4.4 Communication between the AdminServer and ICortexServiceMonitor

The communication between these two components takes place through "Hazelcast" over distributed queues. Every monitored server pulls a queue identified by the server-type, so servers of the same type work on the same queue. The ServiceMonitor implementations can now write their results into this queue. Keep in mind, that the implementations are "dumb" by the means of that they do not interpret the measurements. This applies to the AdminServer. The AdminServer has access of the queues and is there to be registered as an EventListener. As soon as a new measurement is entered into one of the observed queues, the AdminServer registers the corresponding event, evaluates it and refreshes its statistics.

## 4.5 REST Endpoints

The URL prefix for all REST endpoints implemented by the admin server is `http://<host>:8070/admin/res`

### 4.5.1 Cluster members

Returns a list of all members belonging to the same cluster as the admin server.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/clustermembers		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

The list of clustermembers.

```
{
  "clustermembers": [
    {
      "host": "ISRBSNB305.isr.local",
      "port": 5701
    }
  ]
}
```

Listing 4.1: Clustermembers

### 4.5.2 Registered servers

Returns a list of all servers in a cortex environment that are registered by the admin server.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/servers		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

The list of all registered servers.

```
{
  "servers": {
    "BINARYPROXY": [
      "CortexFtpServer"
    ],
    "PULLINGINGEST": [
      "PullingIngestServer"
    ],
    "SEARCH": [
      "SolrSearchServer"
    ],
    "ADMINSERVER": [
      "AdminServer, http://localhost:8070"
    ],
    "STORAGE": [
      "StorageServer, http://localhost:8182"
    ],
    "CS": [
      "CSServer, http://localhost:9998"
    ],
    "TRANSFORMER": [
      "TransformerServer, http://localhost:9997"
    ],
    "WEBSERVER": [
```

```
        "WebServer , localhost:8080"
    ],
    "MESSAGEINGEST": [
        "MessageIngestServer"
    ],
    "MODELSTORE": [
        "TripleStoreServer , http://localhost:8484"
    ]
}
}
```

Listing 4.2: Registered servers

### 4.5.3 Monitoring data

Returns a list of the received monitoring data for a given service type.

**Request:** *HTTP request definition*

Method	Endpoint	Input	Output
GET	/monitor/{servicetype}/data		application/json

**Codes:** *HTTP status codes*

200 all ok  
500 server error

**Output:** *Response data overview*

A map of the received monitoring data, sorted by the id of the responsible service.

```
{
  "data": {
    "4": [
      {
        "empty": false ,
        "comment": null ,
        "serverType": "INGESTPROXY" ,
        "objectSize": 757 ,
        "monitorId": 4 ,
        "serviceType": "INGEST" ,
        "cpuTime": 859 ,
        "systemTime": 1656 ,
        "userTime": 843 ,

```

```
    "totalMemory": 36646912,
    "usedMemory": 21420880,
    "event": null,
    "subData": [
      {
        "size": 757,
        "measureType": "OBJECT_MEASURE"
      }
    ],
    "submitTime": 1307440382914
  },
  {
    "empty": false,
    "comment": null,
    "serverType": "INGESTPROXY",
    "objectSize": 1043,
    "monitorId": 4,
    "serviceType": "INGEST",
    "cpuTime": 171,
    "systemTime": 172,
    "userTime": 156,
    "totalMemory": 36646912,
    "usedMemory": 22087632,
    "event": null,
    "subData": [
      {
        "size": 1043,
        "measureType": "OBJECT_MEASURE"
      }
    ],
    "submitTime": 1307440383602
  }
]
}
```

Listing 4.3: Registered servers



## 5 TestSuiteGenerator

### 5.1 Problem

In order to ensure proper functioning of the DDB, JUnit tests are used to test the correctness of the source code. One JUnit test is used to test one class. In order to test subprojects or even the whole project, so called "test suites" are used. Those can merge Unit Tests and run them all sequential in a given order. That way to test code shall ensure that earlier implemented functionality still works after changes in the source Code. However, test suites need regular maintenance. Each new test has to be added to a suite, in order to keep them up to date. Lacks in communication or forgetfulness cause errors and result in test suites which are out of date. Because of this, test suites become unusable or important tests are never being tested, because they are not included.

### 5.2 To-Be-concept

In order to implement a proper solution for the problem described in the previous chapter, an algorithm that automatically generates and updates test suites shall be developed. That shall minimize the need of maintenance time. That algorithm has to find all tests belonging to a project and automatically generate Java-Code, which is run able and complete without additional work of the developer. However, the developer still must have options to create individual algorithms which create test suites as he needs them. To ensure this, the following functionality has to be included:

- Scan the complete workspace
- Scan single projects
- Exclude single projects
- Rename auto-generated test suites.
- Set storage path for auto-generated suites
- Create hierarchy structures and edit them

The algorithm needs to:

- Find tests
- Generate tree hierarchies
- Generate run able, accurate code

## 5.3 Implementation

The Implementation is stored in an own project called `TestSuiteGenerator`. The classes which implement the algorithms are stored in the package `de.fhg.iais.testsuitegenerator`. Following, those classes are described and explained:

### 5.3.1 TestSet.class

The class `TestSet` creates objects which represent a test suite. A test suite normally consists of the test-classes or sub test suites that have to be executed and corresponding imports. The other components are identical in each test suite, except another test runner shall be used. Because of this, the `TestSets` consists of 3 lists: One containing the names of the classes, one containing the paths for the needed imports, and the last one contains `TestSets` again. That last list is needed to represent and generate a tree hierarchy. The `TestSet` class provides non-static methods which are necessary to generate test suites.

#### **CreateSuite(String destinationPath)**

This method is used to create the java-code of one test suite and stores the .java file in the path, given in the method heads parameter. This test suite includes the tests stored in the `TestSet` and references to stored test suites.

#### **CreateSuiteTree(String destinationPath)**

This method executes the `createSuite`-Method for itself and for all sub-`TestSets`. That creates a test suite for each subsidiary `TestSet`, so that the represented hierarchy is being generated completely.

#### **ConcatTree()**

`ConcatTree()` returns a new created `TestSet`, which includes all tests of the complete tree hierarchy, but no test suites. So the result is a suite containing only the leaves of the tree without hierarchy. The algorithm for this operation is based on a deep search, implemented by a stack.

### 5.3.2 TestSuiteGenerator.class

This class contains all tools necessary for creation of the suites. Following, most important methods are described:

#### **readProject(String name) / readWorkspace()**

Those methods call a recursive algorithm, which is performing a search for tests either for one single project or for the whole workspace. A test is always searched on file system level, that means the folder structure in the project / workspace is being analyzed. An approach using Java reflections was not possible (see chapter 5.1). To return all discovered tests, the instances of TestSet are being used. The readProject-Method returns one TestSet containing discovered tests, readWorkspace() contains one TestSet including sub-TestSets, one for each project folder.

#### **searchTests()**

SearchTests() is declared private and is called by the readMethods(). It reads in all files that a folder contains tries to identify them as test. It analyses a file for the annotation @Test which is an indicator for a JUnit 4 Tests, and for the pattern `extend TestCase` which is necessary for identification of JUnit3 tests.

#### **getWorkspacePath()**

Determines the absolute workspace path currently loaded in eclipse and returns it.

#### **addExcludedFolder(String folderName)**

Can be used to exclude and folder form the whole workspace-search. The folders `.metadata`, `__repository` and `TestSuiteGenerator` are excluded by default.

## 5.4 Usage

In order to use the TestSuiteGenerator, creator-classes have to be implemented. Those classes contain either a main-Method and are executable, or are declared as JUnitTests. By default two creator classes already exist in the project:

### 5.4.1 AllTestsCreator.class

This class creates one suite containing all tests in the whole workspace. This class is a JUnit test, which checks whether a test suite like this already exists and compares the current TestSet with that preexisting suite. Has the amount of tests changed, the test gets red to indicate, that new tests have been written and included, otherwise the test will be passed.

### **5.4.2 HierarchyTests.class**

Creates one suite for each project stored in the workspace, as well as an AllTestsHierarchy-Suite, containing all created project-based test-suites.

## **5.5 Problems**

### **5.5.1 Java reflections**

Another approach for finding tests was the usage of Java Reflections. Using reflections, annotations and methods of a class can be figured out immediately, and therefore ensure an accurate identification of tests. However, reflections can only detect the class itself, not the package structure / hierarchy. Because of that, neither necessary imports can't be included into the suite nor hierarchy test suites can be created.

### **5.5.2 Automatic execution after generating the test suites**

The most efficient way for testing is executing the generated test suite immediately after its creation. That is being hindered extremely with JUnit4, because in contrast to JUnit3 it does not include an graphical testRunner. For executing a test or test suite, a corresponding testRunner has to be executed. That only works within the GUI of eclipse because graphical JUnit4 testRunnter only exist completely integrated in the IDEs like Eclipse. Unfortunately there is no way to access these components, which makes it impossible to execute those runners within the code. Masking the generator as dummy-test suite and compiling the suite within runtime, didn't take any effect, because the Eclipse-plugin seems to cache the tests that shall be run in a special way.

## 6 Native Content

**Purpose** Ingest of content, that are created on the platform. These can be ie. About Pages, News, Virtual Exhibitions.

### 6.1 Mode of Operation

RestCallerServlet.storeSip(HttpServletRequest request)

1. Read form data from request

2. transformation into (internal) DC:

```
String dcXml = createDcMetadataXmlFromRequestParams(rqp);
```

3. creating the SIP, insert the metadata

a) Loading an empty SIP template: `String sip = createSip(request, rqp, dc);`

Internally the following command is executed:

```
SimpleSipCreator sc =  
    new SimpleSipCreator  
    ("resourcetemplates/SimpleSipCreatorTemplate.xml");
```

b) template contains placeholders that have to be filled, ie.:

```
sc.set("ingest-id", ingestId);  
sc.set("item-label", description);  
sc.set("source-metadata", dcXml);
```

4. transformation in the SIP, creating the CRM in SIP

a) loading of the CRM transformers (`NativeContentCrmTransformer.xml`) for the native metadata:

```
String transformer =  
    Home.getResourceLocationAsAbsolutePath  
    (transformers/NativeContentCrmTransformer.xsl);
```

b) processing of the SIP with the transformer:

```
String transformedSip =  
    new XmlStringTransformer(transformer).processXml(sip);
```

5. creating the ingest clients and saving of the SIP (remotely on the client):

```
new Ingest().store(transformedSip);
```

## 7 Quality Assurance of Java Code in the Cortex Platform

### 7.1 Status

- 0.9 19.04.2011 rb: erste Version aus den verschiedenen .txt-Dateien erstellt
- 1.0 28.04.2011 kst, rb: Ueberarbeitung
- 1.1 05.05.2011 rb: Logging-Kapitel hinzugefuegt,  
Sonar-Definition 1.Fassung hinzugefuegt
- 1.2 06.05.2011 cw: MikTex und Variablenamen-Item hinzugefuegt
- 1.3 11.05.2011 tj: translation and integration
- 1.4 10.06.2011 fgub: Ueberarbeitung

### 7.2 Introduction

This document describes all measures taken to assure the quality of the code for the Cortex platform. The Cortex Platform forms the technical basis of the *Deutsche Digitale Bibliothek (DDB)*. The reader is supposed to know the current version of the *Cortex Architekturdokumentation* and the *Grobkonzept der DDB*.

Whoever is interested in the technical "philosophy" behind this document, is recommended the following two books:

- Joshua Bloch, *Effective Java - 2nd Edition*
- Robert Martin, *Clean Code*

### 7.3 Goals and category groups for quality ensuring measurements

The implementation of the Cortex Platform should be

**Z1** correct

**Z2** comprehensible

**Z3** easily changeable

This can not be proofed automatically within the implementation, but in order to approach this ideals, a multitude of measures described in this text are necessary. Most of the measures apply to the classes implemented in Java.

Quality assurance is a task that constantly has to be adepated to altering preconditions - so is this text. Discussions and proposals for change are welcome.

The measures described in the following sections can be divided into five parts:

- B1** Reuse of good frameworks (especially open-source)
- B2** Good structure in the large-scale (by distribution and layering)
- B3** Good packaging of Java-classes (following the design by contract principle)
- B4** Test-driven development (both in Unit and System tests)
- B5** Good structure in the small-scale (by programming-conventions)

## **7.4 Organization of the Eclipse Workspace**

All developers work on a basis of *Eclipse 3.6.2*. All workspaces are homogenous, which means they...

- have the same preferences
- use the same svn-repository
- use the same plugins in the same version
- contain the same projects

### **7.4.1 Required Eclipse Plugins**

#### **Subversive Eclipse Plugin**

The plugin can be loaded from the update site:

http:  
`//community.polarion.com/projects/subversive/download/eclipse/2.0/update-site/`

#### **Maven Eclipse Plugin**

As a prerequisite, Maven 2.2.1 has to be installed on the computer. This installation then has to be used in Eclipse.

The Maven-plugin can be loaded from the update site:

`http://m2eclipse.sonatype.org/sites/m2e`

Just unzip the folder somewhere and set 2 environment variables: `JAVA_HOME` and `M2_HOME`, e.g. `C:\ProgramFiles\Java\jdk1.6.0_25` and `C:\development\apache-maven-2.2.1`. After that you have to add both bin dirs to the `PATH` variable e.g. `;%JAVA_HOME%\bin;%M2_HOME%\bin`

As the last step you should set your maven repository to a different location, because by default it is in your user directory which might be on a network share and that might slow down eclipse later. So create a directory e.g. `C:\development\mavenRepo`, then go to your maven installation directory, open folder `conf`, open the `settings.xml`, uncomment the `localRepository` tag and set it to the folder you just created e.g. `<localRepository>C:\development\mavenRepo</localRepository>`

After restarting Eclipse, the newly installed external Maven can be added and selected by navigating to `Window > Preferences > Maven > Installations`. The Maven instance, provided by Eclipse has proved to be unusable for our purpose.

In general, the Eclipse-Maven plugin does not seem to be optimal. Occasionally build errors are too inconspicuous when starting the build-process from Eclipse. Hence a build should be periodically started from the commandline (`sh`, `cygwin`, `cmd`, ...). This can be done by using the script `makeall`, which is stored in the project `Cortex.Build`.

## Checkstyle Eclipse Plugin

This plugin has proved to be optional as it is also utilized by Sonar (see Sonar for examination of rule violations [7.11]), but often it can be very helpful to be reminded on writing Javadocs(see DBC: Design By Contract [7.8]) by the plugin.

If you do not follow the exact instruction set, you may run into problems with Eclipse:

- Select the update-site: `http://eclipse-cs.sf.net/update`
- open the tree-node and select only the checkstyle-plugin in version 5. Do **not** select anything more, not even the migrations-plugin
- install as usual, restart eclipse
- open `Window > Preferences > Checkstyle`
- create a new profile `cortex-minimal` with `New`
- click the `Import` button and select `Configurations/cortex-minimal-checkstyle.xml` as a file (you find these files in the svn repository, Cortex -i documents -i QA-Configuration)
- make this the `Default` profile

Select all projects in the package-explorer, select `Activate Checkstyle` in the context-menu. Every violated rule is attended by a magnifying glass and a warning. So far only two rules are set up for a good writing style:

- Javadoc for public, non-property methods.
- Naming conventions

To view the rules, follow this guidance:

- within the XML Document itself. This is inconvenient, though of course quite possible.
- Best practice for reviewing single rules (Attention!)
  1. open `Window > Preferences > Checkstyle`
  2. double click `cortex-minimal-checkstyle`
  3. open `treetag javadoc comments`
  4. click, but do not double-click the tag: `method javadoc`. Double-clicking anyway activates the rule a second time, but this time makes it a default (producing far too many warnings). So you better undo the second activation.
  5. on the right-hand-side, `method javadoc` now shows up as an enabled module
  6. double click this entry to view it's settings
  7. the question mark on the right offers more info

There should always be a discussion on a developer meeting before activating or customizing more rules. In most of all cases it is a consensus to avoid a bigger numbers of warnings, as too many warnings tend to interfere with the development work.

## TeXclipse Eclipse Plugin

This plugin is optional too, but facilitates the editing of `.tex` files significantly and is a premise for creation of PDF-files directly from Eclipse. The installation of a LaTeX-compiler like MikTeX (<http://miktex.org/>) is a required prerequisite. The installation itself is trivial and documented elsewhere.

To activate LaTeX support in Eclipse, you have to install the TeXclipse plugin from the update site <http://texlipse.sourceforge.net>. After restarting Eclipse, the plugin has to be configured. In `Window > Preferences > TeXlipse > Builder Settings`, you need to enter the `/bin` directory of the MikTeX-installation in the area: `Bin directory of TeX distribution`. The programs and scripts of the installation are recognized automatically. Next, the PDF-viewer has to be configured under `Viewer Settings`. Enter the viewer of your choice in the area `Edit`.

## 7.4.2 Settings in Eclipse

### UTF-8

In `Window > Preferences > General > Workspace`, you need to change the encoding to UTF-8.

**Note:** Encoding problems are not only annoying, but can also lead to subtle errors. Imported files often are encoded otherwise, ie. latin-1. You can find out about the encoding of a file like this:

- \*nix: type: `"file filename"`. To change the encoding, type `"recode utf-8 filename"`. For more information, consult the `--help` option or the info pages.
- Firefox: after removing the suffix, see `View > Character Encoding`
- gvim: `":set fileencoding:"`, to change `":set fileencoding=utf-8"`

### Java Editor Code Style Formatter

Import the formatting file `Configurations/cortex-compact.xml` from `Window > Preferences > Java > Code Style > Formatter`. This file determines the java sources and is preconfigured for a clearly laid out, compact format. For documentation purposes, this XML file is supplied in the workspace itself.

### Java Editor Editor Save Actions

The actions, that are executed on every writing of Java-sources can be set under `Window > Preferences > Java > Editor > Save Actions`. These actions simplify the comparison between different revisions of a Java-source and the overview. The following checkboxes have to be adjusted on the `Save Actions` page:

- Format source code *ALL* lines
- organize imports

And in addition (the order between the Eclipse view and the displayed text are is a little different in comparison to the tabs, where the actions can be defined):

- Remove trailing white spaces on all lines
- Use Blocks / Convert control statement bodies to block
- Use this qualifier for field accesses always
- static accesses use declaring class change ... through subtypestype
- static accesses use declaring class change ... through instances
- Add missing '@Override' annotations
- Add missing '@Deprecated' annotations

- Remove unused imports
- Remove unnecessary casts
- Remove unnecessary '*NON – NLS*' tags

### HTML Editor

See **WEB > HTML files > Editor** for the following adjustments:

- line width 110
- uncheck all checkboxes, except for **align final brackets**
- select radiobutton **insert spaces** (not tabs)
- indentation 4
- check content assist and grammar constraints

### XML Editor Preferences

Under **XML > XML files > Editor** set:

- line width 110
- uncheck all checkboxes, except for **insert white spaces before closing**
- select radiobutton **insert spaces** (not tabs)
- indentation 4
- check content assist and grammar inferred grammar

### Java Code-Templates

For the time being, there exists only one template: **LOG**. This template can be used in the Java editor. After typing **LO**, selecting the completion **<ctrl>+<space>** and then choosing the **Log**-template, you get the typical declaration of a static logger and its standard initialization. New templates are welcome, should however be discussed by the team first.

Templates are loaded by importing the file **Configurations/cortex-templates.xml** from **Window > Preferences > Java > Editor > Templates**. For documentation purposes, this XML file is supplied in the workspace itself.

## Configuring SVN

Within the SVN repository exist a trunk, branch and tag for all projects as proposed by the SVN specification. This yields the least problems. The SVN repository is available under

`http://nm-ddb-data.iais.fraunhofer.de/svn/platform`

In the directory "development", you can find trunk, branches and tags. You need all the projects from there.

In case of removal or adding projects, a team-project-file (\*.psf) is sent via email. By importing this file, all the projects in the workspace are adopted to the condition in the repository.

## Java Compiler Compliance Level

Set the compliance level 1.6 under Window > Preferences > Java > Compiler > Compliance. Be sure that all projects use this value, that is, in no projects under <Project-Name> > Context-Menu > Properties > Java Compiler the value `enable project specific setting` should be set.

## Restart Eclipse

Sometimes Eclipse feels like hanging up might be a good solution to problems and then nothing you just changed would be saved. So better restart Eclipse now.

## Eclipse Settings/Tuning

You should point your workspace to a local directory on your computer instead of a network drive, this improves the performance of eclipse.

Sometimes Eclipse has problems finding the correct JVM, you can counter that by setting the -vm option in the eclipse.ini to your JDK javaw.exe path:

```
-vm
C:/ProgramFiles/Java/jdk1.6.0_25/bin/javaw.exe
```

It is important that you put the -vm option above the -vmargs and that -vm and the path are in two different lines.

By default Eclipse starts with little memory and it can also be tweaked for better performance. You should atleast set the -Xmx and XX:MaxPermSize parameters in the eclipse.ini (eclipse installation directory) and if youre interested in some more performance tweaks follow this link:

`http://stackoverflow.com/questions/142357/what-are-the-best-jvm-settings-for-eclipse`

## 7.5 Organization of the projects in the Eclipse-workspace

Everything under development (ie. Java-classes, XSL-transformations or property files) is combined in the Eclipse workspace. Every reference (ie. open source frameworks) is declared in the same Eclipse workspace, including the version number. The complete workspace is synchronized with the svn-based repository. For a developer who has committed his changes it is possible to delete his workspace at any time. If he checks out all projects from the repository, he gets a fresh, functioning and up-to-date workspace.

Derived files (like .class files) are to be kept out of the repository. If there are files that have to be customized (ie. property files as they contain path-declarations), these files should not be committed into the repository. The same rule applies to ingested data.

For all DDB-specific configuration (ie. the used harmonized model from CRM, EDM, FRBroo, DC, the SolR configuration, vocabularies), a separate location has yet to be defined. So far these configurations and testdata are still stored in the repository.

Cortex is a distributed system of servers. Each of these servers is designed in tiers. The projects reflect this structure. Every project depends on

**other Cortex-projects:** This reflects the tiers from upside-down. As Eclipse does not allow cyclic dependencies of projects, the correct layering is guaranteed. The projects should be of similar size. Otherwise the layering could be irrelevant as single layers contain almost all the classes and the other layers deliver only insignificant parts.

**external projects:** (ie. open-source-frameworks). You have to keep in mind, that the scope of project matches the one of the external project. A utilities-project for example should not depend on a REST-framework - this dependency would rather be expected in a high layer.

All internal and external dependencies are described with assistance of Maven:

- Every project declares the internal and external projects on which it depends in a `pom.xml`-file, located in the projects-folder. This description should be as compact as possible and does not specify version numbers - in particular not for external dependencies. In every projects-file, the `pom.xml` of the project `Cortex_Build` must be declared as a parent.
- the parent-project `Cortex_Build` contains the complete list of all dependencies, including the required version numbers in the `dependencyManagement`-section.

You will have to pay attention, that in each project only the most essential dependencies are declared. Unused dependencies have to be removed just like code that is unused or commented out. Centric declaration of the versions helps avoiding the clutter, that otherwise occurs in case every project may define the version of an external project all by itself.

Utilizing Maven along with the Eclipse-maven-plugin produces a state where Cortex can be built in two different ways:

- via Eclipse, analyzing the dependencies with assistance of Maven's `pom.xml`
- via shell, in which Maven can be executed directly. Building Cortex this way is used in the various scripts in the project `Cortex_Build`

Because of the deficiencies of the Eclipse-Maven Plugins, there can be subtle differences between these two ways of building the Cortex - although they are considered as insignificant.

## 7.6 Using SVN

*Never commit without comment.* Comments have to be phrased as precisely as possible. Only compilable and tested code may be committed. If this turns out to be impossible (due to handover of partial results to other developers), a mail must be sent to the entire development team. All projects should be committed collectively.

The following approach has proved to be useful (step by step):

- select all projects in the package-explorer
- in the context-menu, select **Team > Synchronize**
- then change to synchronize-view (automatically...)
- press the **[+]** (Expand all) button
- solve all conflicts
- fetch updates
- commit your own changes

In the synchronize view, you can see the count of the conflicts/incoming/outgoings on the lower right-hand-side.

1. *Conflicts* have to be solved manually and individually for every resource:

- discard your own changes: **Overwrite and update**, **Revert**
- unusual: reject changes from the repository: **Overwrite and commit**
- otherwise: merge the changes per resource, usually in the diff-view. You can jump delta-wise, apply, edit and by selecting **Mark as Merged**, declare the merged version in your own workspace as the relevant one.

By doing this, your merged version becomes part of the committable resources (outgoings)

2. *Changes* from the repository must be updated, reviewed for errors and tested
3. *Commits* finally are the own changes in groups with the essential comments

In case of structural reworks, like renaming or deleting multiple resources, especially folders, it is common practice to select all projects in the package-explorer and selecting **Team > Update** in the context-menu. This enables the resources atop of the committed ones, that have been changed indirectly and have a higher revision number in the repository than in the Eclipse-workspace, to obtain the revision number of the repository.

If you do not do this, this might result in conflicts later on. Just changing only existing resources should be no problem.

For the property `svn:ignore`, there should be defined a projects-wide, global value, so that no inconsistencies can emerge when synchronizing with the SVN repository. As we have not determined any problems in this area in the past, no appointment have been made so far.

## 7.7 Test-driven development

This section discusses primarily Unit-tests, only the last section covers System-tests (integration-tests).

### 7.7.1 Purpose

It is a defined goal to support software development with Unit-tests. The team thinks, that even just writing tests itself leads to better software because:

- the class under development is reviewed from the inside (developer) and from the outside (tester): The developer concentrates on the "how do I build this", the tester on "what can I do with it (why do I build it)". This also applies when - as in our case - developer and tester are the same person.
- the interface of the class improves: if the setup of the test is too difficult, most likely the context is too large.

Writing test classes does not slow down the development of software as the effort is merely shifted from debugging to writing tests. If a developer does not write a test for his class, there should be a better excuse than "I ran out of time".

### 7.7.2 approach

A Test-class normally covers exactly one class. Different classes have different Test-classes. Tests are implemented in JUnit4 Syntax.

- the Test-class should be named `ClassToBeTestedTest`.

- the class to be tested is in a project in `src/main/java`, the test-class in the same project in `src/test/java` in the same package.
- when building the project (*assemblies*), the test-classes are not delivered.

Test-classes are always stand-alone, thus can be executed separately. Simultaneously they are integrated into Test-suites, so that larger arrays can be tested at once. In case they use a text-external resource, like a running server, it is recommended to describe this in the test's Javadoc. Maven simplifies the work with tests of a project by automatically selecting and executing all tests from the source-folder (`src/test/java`). This is used by the CI (continuous integration) (see CI: Continuous Integration with Hudson and Sonar [7.10])

### 7.7.3 Types of Unit-tests

We distinguish between several types of unit-tests. You should be aware that this distinction is not precise and you could allocate a test to different categories.

#### Tests for development steps

The developer secures the stepwise implementation of his class during first-time development. Intentions of these tests are:

- meaningful division of a class into several smaller functions
- resolve uncertainties about the behaviour of used classes (also from `rt.jar`)

There is no point in removing the tests afterwards. Whenever the development of the class is continued, these tests surely are of good use. Even if it's just for breaking in to the old code again. If they turn out to be useless during the development process, they can be removed by another developer later on.

#### Semantic-tests with little effort on setup

These tests form the largest amount of the important tests as they focus on guaranteeing the interface to be good, minimal, easy to supply and the implementation is as well free of errors. Other developers can use the test as a reference implementation of the class ("this is how the use of this class is intended").

#### Semantic-Tests with large effort on setup

Most of all, this applies to access on persistent storage solutions, starting/stopping a server, etc.. These tests tend to have a longer duration. You should not recoil from developing methods

that imply a setup and contribution of testdata. Once developed, you gain a lot of quality in the system.

You can avoid overhead quite often by splitting up the implementation in the matter of

- ease of testability of the methods: Parameters, that have to be obtained in a complex manor should be provided as a parameter to the methods in advance.
- few, short methods with a trivial flow of control remain providing parameters. For these, either sophisticated tests can be written or - more commonly - the code is validated through inspections (documented in the methods)

This structure reflects the concept of the "humble objects" (Martin Fowler).

### **Testing process**

Tests are executed periodically by the developer during his work. Before the commit, relevant tests, but at least the unit tests of the altered projects, should have passed with a positive result. Furthermore you have to guarantee, that the ingest of the SIPs (specifically the ingest of digital objects into the Cortex platform) works correctly. For this, the Test-suite "MinimalTest" containing the corresponding tests is located in the project `CS_IntegrationTest`. Because of the dimension of the tests, it is not advised to run all the tests prior to a commit as this is anyway continuously done within the scope of Hudson.

Requirements to a minimal test coverage have not yet finally been specified. Nevertheless it is vital to improve the test coverage.

### **System-Tests**

Just like Unit-tests, System-tests have to run automatically. All system-tests are stored in the project `CS_IntegrationTest`, as their configuration overhead is generally of a larger scale.

Functional System-tests are written in JUnit4, Load-tests with the help of JMeter.

Located in the package `de.fhg.iais.cortex` of the project `CS_IntegrationTest`, you can find the test-class `FullCortexTest`. This provides a frame for tests of the `ASC Transform`, `Pulling Ingest`, the `WebInterface (WebsiteTest)` and the `REST Endpoints (RestInterfacesTest)`, and it works alongside the existing repository directory.

During the development, modularity was paid lots of attention to, so that sub-tests with a running Cortex-server are able to run on their own.

## **7.8 DBC: Design By Contract**

One of the most effective methods in the outline of good software is DBC (design by contract). Originally made popular by Bertrand Meyer in connection with the language Eiffel,

there exist many variants, fit to many different programming languages. The Wikipedia-article [http://de.wikipedia.org/wiki/Design\\_by\\_contract](http://de.wikipedia.org/wiki/Design_by_contract) is not wrong at all but advised only with restrictions, because of the core-statement "how do we produce good software?" is too loosely defined.

### 7.8.1 Contracts or Agreements

Whether a contract is good or bad can not be verified by syntax. Syntactic verifications can merely be an indicator, suggesting existing problems. This is the elemental boundary of otherwise powerful tools like Sonar: they deliver a multitude of suggestions to improve or inspect the code, but even when they do not display any more warnings, you cannot make a point about the quality of your software, that is devised by the quality of the contracts.

In the terms of DBC, methods are developed. A method is characterized by a contract between a client (caller), and a supplier (callee).

As a satisfying concept for a formal definition of contracts does not exist at all, even for innovating projects like Cortex, only the colloquial verbalization remains.

Contracts are written with Javadoc into the method's head of the supplier. By Using generators, you are enabled to produce a complete documentation of contracts just like Integrated Development Environments (IDE) can render these documentations and provide them as a tooltip dynamically. Contracts in Javadoc are written once and read a thousand times. During development of a class and its methods, just like when implementing the testclass, these contracts are present in the heads of the developers. So they have to be documented and formulated carefully and correctly.

A contract must at first be framed as an open sentence, describing the function of the method:

```
/**
 * loads an entity from the DB.
 * clazz provides the DB-table, from which to be loaded,
 * id provides the key for the line to be searched.
 * A row with a matching key has to be present, otherwise
 * a RuntimeException is thrown
 *
 * @param <T> result type parameter
 * @param clazz class of the entity to be loaded, must not be null
 * @param id key of the entity to be loaded, must not be null
 * @return entity, not null
 */
<T> T load(Class<T> clazz, Serializable id) {
```

- parameter requirements are fixed (here: "not null" in the `@param`). Requirements may also be located in the textblock at the beginning for better readability.
- expected results are fixed (here: "not null" in the `@return` statement). Requirements may also be located in the textblock at the beginning for better readability.
- "load" obviously assumes the client to know there's a row with the given key. Under this premise the implementation of `load` can fulfill its contract.
- contracts should always be phrased as precise as possible. As a rule of thumb: a method that doesn't require anything cannot deliver anything safely. This normally results in massive amounts of "`!= null`"-tests.
- only at the system's edge: Exceptions are to be caught and just like a failed assertion lead to a notification instead of an exception. In case the system's edge is a REST-endpoint, the matching http return codes must be defined carefully.

After the contract is worded in common speech, it has to be verified during runtime. This redundancy is quite unpleasant, but here, too, Java does not deliver an appropriate concept for improvement, than by the following "best practice": contract violations must lead to runtime exceptions. For this scenario, there exists the class `DbcException`, a subclass of `RuntimeException`.

In the class `Assert` a multitude of tests are implemented, helping on the verification of the contracts. The most important are `isTrue(...)` and `notNull(...)`. The class may be extended when necessary. A contract assertion then looks like this: `Assert.notNull(clazz);`. The verifying method throws the exception. An `if`-statement with an embedded `throw new DbcException(\ldots)` is acceptable, too.

## 7.8.2 Handling exceptions

DBC presumes that every method call implies a contract (even if it is undocumented, in this case just nobody knows it). So far, a contract is violated in most cases by an unsatisfied precondition. Contract violations lead to exceptions.

These exceptions should be caught and compensated on the respective edges of the system or subsystems. Otherwise the problematic to erroneous situation must be reported to the "outside". Typical for this scenario is the `Ingestor` in `Cortex`. It is a component of the system's edge. It may utilize one or many `try-catch`-blocks to handle exceptions from the inner of the ingest-process and transform them into error messages, loggings and reports.

In the inner parts of a system or subsystem, a compensation of an error is only scarcely possible. Normally exceptions are just passed along, sometimes (because of distrust in the client) logged and handed over. Because of the high probability of the handover, only `RuntimeExceptions` should be thrown (standard: `DbcException`). The class `RuntimeException` itself should not be

used, own derivatives are acceptable, though. In case a checked exception is caught but cannot be compensated, a `DbcException` is thrown, containing the caught `Throwable` as a 'cause'.

In addition, runtimeException-classes from `rt.jar` are used:

- `NullPointerException` - Reference is null
- `IndexOutOfBoundsException` - Index value is out of range
- `ConcurrentModificationException` - Concurrent modification of object has been detected despite prohibition
- `UnsupportedOperationException` - Object does not support method

## 7.9 Logging in production-, and testsystems

### 7.9.1 Purpose

Logging has the following purposes:

- A.** delivering information to trace the process and errors during development time.
- B.** delivering information to trace errors and analyze the system's behaviour during runtime.

Accordingly there are log-levels, that should be used consistently within the project (please keep in mind, that there is *no* "fatal!"):

TRACE	for <b>A</b>	very detailed information in development rather to be avoided
DEBUG	for <b>A</b>	detailed information for development and for analyzing system behaviour
INFO	for <b>B</b>	analyzing system behaviour
WARN	for <b>B</b>	potential misbehaviour, "overload, ..."
ERROR	for <b>B</b>	Error, that has to be analyzed, usually by stacktrace

At relevant spots in the the source code a LOG-statement is placed, ie. `LOG.warn(''\ldots text \ldots'');`. The warning message is displayed, if the log level is less or equal - so if the log-level is TRACE, DEBUG, INFO or WARN, thus below the level specification in the statement. During development, the application is usually run on DEBUG-level, sometimes on INFO, only rarely on WARN. During operation usually INFO is used, occasionally WARN. The logging level of the application can be set differently for individual classes / components.

## 7.9.2 Slf4j as a façade for logging

The main problem in logging is that there are several logging frameworks. This results in a state where the different frameworks that are utilized in Cortex use a different logging framework respectively. Then again, the logging should be configured once and for all, the outputs should be formatted in the same manor and be evaluable easily.

In it's own implementations, Cortex uses the logging façade `slf4j`. During runtime of the application, a logging framework needs to be configured as an implementation. In Cortex, `logback` (a successor to `log4j`) is used. For the time being, the following frameworks are still in use by the utilized frameworks:

- `java.util.logging`
- `commons-logging`
- `log4j`

For all the different logging frameworks exist bridges in order to reroute their logging to `slf4j`, so that for Cortex *only* `slf4j` needs to be configured solely. It is important to make sure, that *before* the first logging, the following code-sequence is executed over JUL (see also `de.fhg.iais.cortex.Starter`).

```
// SLF4J is bound to logback
LoggerContext lc = (LoggerContext) LoggerFactory.getILoggerFactory();

try {
    JoranConfigurator configurator = new JoranConfigurator();
    configurator.setContext(lc);
    lc.reset();
    configurator.doConfigure("conf/logback.xml");
} catch ( JoranException je ) {
    je.printStackTrace();
}
StatusPrinter.printInCaseOfErrorsOrWarnings(lc);

LOG.info("-- CORTEX starting");

java.util.logging.Logger rootLogger =
    LogManager.getLogManager().getLogger("");
Handler[] handlers = rootLogger.getHandlers();
for ( int i = 0; i < handlers.length; i++ ) {
```

```
    rootLogger.removeHandler(handlers[i]);  
}  
SLF4JBridgeHandler.install();
```

### 7.9.3 Configuration

The configuration is handled in `Cortex/conf/logback.xml`. Who is writing the output data and which `<pattern>` determines the output format is located in the `<appender>`. The `<root>`-Element declares the default logging level for all loggers. Modifications of the default-log-level are configured by `<logger>`-entries:

```
<configuration>  
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  
    <encoder>  
      <!-- <pattern>%d{ISO8601} [%thread] %-5level  
                %class.%M:%L - %msg%n</pattern> -->  
      <pattern>%d{ISO8601} %-1level [%thread] - %msg%n  
                %replace(%caller{1}){Caller\+\d\s*, ""}</pattern>  
    </encoder>  
  </appender>  
  
  <root level="warn">  
    <appender-ref ref="STDOUT" />  
  </root>  
  
  <logger name="de.fhg.iais" level="info" />  
  <logger name="de.fhg.iais.asc" level="debug" />  
</configuration>
```

Logger names form (because of the dots in the name) a tree. The most specific `<logger>` definition respectively determines, which level is activated for the logger. For example:

1. all logs above WARN are displayed (WARN and ERROR)
2. except from loggers, whose names start with `de.fhg.iais`. They log above INFO (so INFO, WARN, ERROR)
3. except from loggers, whose names start with `de.fhg.iais.asc`. They log from DEBUG on.

Additional information about the logging framework Simple Logging Facade for Java (SLF4J) can be found on its website <http://www.slf4j.org/>.

### 7.9.4 Declaring loggers in the source-code

Every class utilizing logging mechanisms (which turns out this is all classes ...), declare a private logger-constant. Please use notation (capital letters), modifier (private, static, final) and reference (containing class), just like in this template:

```
private static final Logger LOG = LoggerFactory.getLogger(X.class);
```

If you have loaded the templates described in this text, you can as well use `LOG<CTRL+SPACE>` to expand the Log template. The name of the logger then is `pa.ck.ag.e.name.ClassName`.

### 7.9.5 utilizing Loggers, Best Practices

Logging should be compact. Too much logger-output (ie. a production system with DEBUG log level) tends to decrease the performance of the system dramatically. For debug-purposes, there has to be enough logging to reasonably comprehend the flow-of-control. If the logfile has to be evaluated, you should put into consideration to implement text markers like `[[dc-sip]]` into the logging output in order to better be capable of extracting the relevant data via script. Structured output is useful in every scenario. Relevant data should always be displayed.

String concatenation in the fashion of `"c:" + c + "`, `d:." + d` can be used without problems if the log level of the output is high enough or if the log statement is passed through rarely. Many developers rate it as unelegant (at best), to evaluate the parameters of log statements everytime before the log method itself is called, because many cpu-cycles are wasted in this scenario, in which the logger is inactive though due to the configuration. This can be handled as follows:

- *The* typical usage is :

```
LOG.error("Exception while stopping server", t);
```

- Up to two parameters can be forwarded elegantly and efficiently with the `{}`-Notation. This way is also performant, if the logger is *not* active.

```
LOG.debug("File {} has {} UTF8-chars", fileName, n);
```

- Using more parameters provides some minor costs — even if the logger is inactive — as a parameter-array must be built anyway:

```
LOG.debug("f:{} d:{} owner:{},new Object[]{f,d,user});
```

- Only in methods called very frequently and there, in their inner loops the following idiom pays off:

```
if (LOG.isDebugEnabled()) {  
    LOG.debug("...");  
}
```

- Imaginable, but not recommended, in most extreme situations is the utilization of the Java-compiler as a preprocessor (similar to the preprocessor in C / C++). A constant (static final) allows the Java compiler, not to generate code for an `if`-statement in the first place if the condition is assuredly `false`. The risk to cause damage by this distributed logging mechanism is high.

```
if (DEBUGGING_ENABLED) {  
    LOG.debug("...");  
}
```

- assume, you have a good `toString()`-method in class X, that even may be complex, and x be an object of class X, an elegant *and* efficient logging would be:

```
LOG.debug("x: {}", x);
```

## 7.10 CI: Continuous Integration with Hudson and Sonar

It is of great benefit to plan reviews of parts of the developed software periodically. External partners should be able to obtain read-only access to the SVN-repository of the Cortex platform respectively the DDB in order to check out a current version to run unit-, and integration tests. For the development-team, a *continuous inspection* of the state of the code is of great importance. This analysis has to span these two areas:

**Testing the developed software:** Of course, if the testsuite does not find an error, it does not imply a software to be free of errors. In fact, the testsuite offers the developers a warranty, that their changes do not destroy anything, other developers regarded as important enough to develop testsuites for. Without a good test coverage in a testsuite, the developer's courage to continue the development and improvement of the software fades.

**finding (undesirable) programming patterns:** Even if in Chapter DBC: Design By Contract [7.8] we mentioned, that despite syntactic checks, that mostly apply to (unwanted) programming patterns "on the small scale" do not measure the quality of the product directly, they still are of great use. They are outstanding indicators whether the software in development ever has the chance to reach the goals described in 7.3.

Hudson supports the following tests:

**Test of the developed software:** As the building and test of the Cortex platform currently takes about 10 – 20 minutes, Hudson is configured to run the test on every commit. Hudson downloads the code from the SVN-repository and builds the Cortex-assembly with assistance from Maven, runs the tests and notifies the developers via different channels (mail, reports, feeds) about problems.

**Finding (undesired) programming patterns:** For this task, we utilize the Sonar-plugin in Hudson. These plugins enable Hudson to apply configured syntax-rules on the code and to generate reports afterwards. The webinterface can be used by an administrator to configure the syntax-rules, and by developers to check the results of a check. The webinterface does not only provide an overview of the project, but — besides many other possibilities — also a drill-down according to syntax-rules, to specifically examine rule violations.

## **7.11 Sonar for examination of rule violations**

### **7.11.1 The Sonar rulesets**

The discussion, how big or small the active ruleset may be, is taken in all developer teams. On the one hand, too many displayed rule violations lead to inattentiveness regarding the rule violations. Also, there exists a series of rules, which application is controversial at best (ie. checkstyle "design for extension"). On the other hand, there is a desire for a most-possible detailed view on the parts of Cortex, that potentially portends problems - which is, where more and different indicators are very helpful.

For the time being, rather many rules are activated in the Sonar:

- Checkstyle: 30 rules
- PMD: 85 rules

### **7.11.2 Rule violations in the Cortex code**

The 28.4.2011 Sonar displayed the following result:

- Blocker: 0
- Critical: 18
- Major: 684
- Minor: 509
- Info: 214

Distributed on the different categories, this results to:

- Efficiency: 46
- Maintainability: 785

- Portability: 9
- Reliability: 287
- Usability: 297

Reviewing the errors in the source obviously provides large potential of improvement, but even the critical and serious rule violations weren't those of an alarming kind. Nevertheless there will be a point in time to correct rule violations and to analyze, if the fragments in which they occurred have to be refactored.

## 7.12 Current rules for the small scale quality checks

The following list attracts attention to the more significant rules. *This is not final!* It does not even provide a special order. In an ideal case it would have even been empty . . . .

- javadoc in the sense of DBC (at least) for all non-private methods, that are not pure setter/getters.
- the boy scout's rule: "Leave the campground cleaner than you found it." **TODO: ref:(Clean Code)** . Loosely translated: If you alter a class, improve anything else in it, too - even if it's not much.
- always use generics in declarations, distinguish between interface and implementation:

```
List<Integer> il = new ArrayList<Integer>();
Collection<? extends Collection<String>> a =
    new ArrayList<HashSet<String>>();
```

- narrow down the scope as far as possible. Standard for fields (instance variables) is **private**
- implementing **equals** implies implementing **hashCode**, otherwise there's a high probability that not all collections work as intended.
- when assigning labels, please do not consider technical names (ie."list" for a list). Use subject-specific names respectively.
- always keep in mind: code is written once, but read a thousand times (and changed). Clarity is important: Use **//** as a line break, well-named local variables and no too complex expressions as they are unreadable, scatter in the debug output and interfere in finding **NullPointerExceptions**.
- in terms of readability, a part of the (non-Javadoc!) documentation may be relocated into the code by descriptively labeling variables. These may even be of a longer kind.

The Eclipse CamelCase-autocompletion feature facilitates the typing work noticeably. For example type `fCT` and press `<CTRL>+Space` to expand to `FullCortexTest`. This works for either class-, methods-, or variablenames.

- The developed code belongs to everyone. As a consequence of this "common code ownership": whoever edits a class, has to understand it before. Only then he obtains the right for customizations, but the responsibility to be careful and to use and - if necessary - extend the tests. A grave customization consists of removing warnings, but mind far-reaching effects! This is not filed under the "boy scout's rule"...

## 7.13 Declaration of the Sonar rules

The following XML file has only documentation purposes, it is not intended to be read. In the future, an adjustment of the rules will be necessary. The most current wording at a time always has to be exported from Sonar after every change (by the Sonar-administrator), formatted by an XML tool, and documented both here and in the SVN repository.

```
<?xml version="1.0" encoding="UTF-8"?><!-- Generated by Sonar -->
<profile>
<name>Cortex</name>
<language>java</language>
<rules>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.coding.ParameterAssignmentCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.design.HideUtilityClassConstructorCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.coding.SimplifyBooleanExpressionCheck
</key>
<priority>MAJOR</priority>
```

```
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.naming.StaticVariableNameCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.modifier.ModifierOrderCheck
</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.naming.MethodNameCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.design.FinalClassCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.coding.EmptyStatementCheck
</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.coding.InnerAssignmentCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
```

```
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.coding.DoubleCheckedLockingCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.imports.UnusedImportsCheck
</key>
<priority>INFO</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.coding.StringLiteralEqualityCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.modifier.RedundantModifierCheck
</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.coding.RedundantThrowsCheck
</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.coding.IllegalThrowsCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.sizes.AnonInnerLengthCheck
```

```
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.naming.MemberNameCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.naming.ConstantNameCheck
</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.naming.PackageNameCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.coding.DefaultComesLastCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.coding.EqualsHashCodeCheck
</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.coding.SimplifyBooleanReturnCheck
</key>
<priority>MAJOR</priority>
```

```
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.coding.HiddenFieldCheck
</key>
<priority>MAJOR</priority>
<parameters>
<parameter>
<key>tokens</key>
<value>VARIABLE_DEF</value>
</parameter>
<parameter>
<key>ignoreConstructorParameter</key>
<value>true</value>
</parameter>
<parameter>
<key>ignoreSetter</key>
<value>true</value>
</parameter>
<parameter>
<key>ignoreAbstractMethods</key>
<value>true</value>
</parameter>
</parameters>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.naming.LocalFinalVariableNameCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppycrawl.tools.checkstyle.checks.design.VisibilityModifierCheck
</key>
<priority>MAJOR</priority>
</rule>
</rule>
```

```
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.coding.MagicNumberCheck
</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.naming.LocalVariableNameCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.naming.ParameterNameCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.metrics.BooleanExpressionComplexityCheck
</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>checkstyle</repositoryKey>
<key>com.puppcrawl.tools.checkstyle.checks.metrics.CyclomaticComplexityCheck
</key>
<priority>MAJOR</priority>
<parameters>
<parameter>
<key>max</key>
<value>10</value>
</parameter>
</parameters>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UnusedNullCheckInEquals</key>
```

```
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>StringInstantiation</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>ConstructorCallsOverridableMethod</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidCatchingNPE</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidRethrowingException</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>ReplaceEnumerationWithIterator</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidArrayLoops</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UnusedFormalParameter</key>
<priority>MAJOR</priority>
</rule>
<rule>
```

```
<repositoryKey>pmd</repositoryKey>
<key>EmptySwitchStatements</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>ClassCastExceptionWithToArray</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidThrowingNullPointerException</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UnusedPrivateField</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>CompareObjectsWithEquals</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UseIndexOfChar</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>BigIntegerInstantiation</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>FinalFieldCouldBeStatic</key>
<priority>MINOR</priority>
```

```
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>IfStmtsMustUseBraces</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>SuspiciousEqualsMethodName</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>InstantiationToGetClass</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>SuspiciousHashCodeMethodName</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>LooseCoupling</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UnusedLocalVariable</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UnnecessaryCaseChange</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
```

```
<key>EmptySynchronizedBlock</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>SingularField</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UnusedPrivateMethod</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>CloseResource</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidCatchingThrowable</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>EmptyWhileStmt</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>CollapsibleIfStatements</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UselessOperationOnImmutable</key>
<priority>CRITICAL</priority>
</rule>
```

```
<rule>
<repositoryKey>pmd</repositoryKey>
<key>CloneMethodMustImplementCloneable</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UselessOverridingMethod</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UnusedModifier</key>
<priority>INFO</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>PreserveStackTrace</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UseArraysAsList</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidThrowingRawExceptionTypes</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>EmptyIfStmt</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>EqualsNull</key>
```

```
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>BrokenNullCheck</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UseCorrectExceptionLogging</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>InefficientStringBuffering</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>ArrayIsStoredDirectly</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>FinalizeOverloaded</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>ClassNamingConventions</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>DontImportSun</key>
<priority>MINOR</priority>
</rule>
<rule>
```

```
<repositoryKey>pmd</repositoryKey>
<key>DontImportJavaLang</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>MissingStaticMethodInNonInstantiatableClass</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>StringBufferInstantiationWithChar</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UseArrayListInsteadOfVector</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>WhileLoopsMustUseBraces</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>StringToString</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>SimplifyConditional</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>ReplaceVectorWithList</key>
<priority>MAJOR</priority>
```

```
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>NcssMethodCount</key>
<priority>MAJOR</priority>
<parameters>
<parameter>
<key>minimum</key>
<value>50</value>
</parameter>
</parameters>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidPrintStackTrace</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>MethodWithSameNameAsEnclosingClass</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>SuspiciousConstantFieldName</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>NcssTypeCount</key>
<priority>MAJOR</priority>
<parameters>
<parameter>
<key>minimum</key>
<value>800</value>
</parameter>
</parameters>
</rule>
```

```
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidInstanceofChecksInCatchClause</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>EmptyFinallyBlock</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>IntegerInstantiation</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidDollarSigns</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidAssertAsIdentifier</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UnnecessaryLocalBeforeReturn</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidCallingFinalize</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>BooleanInstantiation</key>
```

```
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UnconditionalIfStatement</key>
<priority>CRITICAL</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>CloneThrowsCloneNotSupportedException</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>ReplaceHashtableWithMap</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidDecimalLiteralsInBigDecimalConstructor</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>ForLoopsMustUseBraces</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>EmptyTryBlock</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>SignatureDeclareThrowsException</key>
<priority>MAJOR</priority>
</rule>
<rule>
```

```
<repositoryKey>pmd</repositoryKey>
<key>EmptyFinalizer</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>IdempotentOperations</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>ExceptionAsFlowControl</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>EmptyStaticInitializer</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>FinalizeDoesNotCallSuperFinalize</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UseStringBufferLength</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidEnumAsIdentifier</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>IfElseStmtsMustUseBraces</key>
<priority>MAJOR</priority>
```

```
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>SystemPrintln</key>
<priority>MAJOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>UselessStringValueOf</key>
<priority>MINOR</priority>
</rule>
<rule>
<repositoryKey>pmd</repositoryKey>
<key>AvoidDuplicateLiterals</key>
<priority>MAJOR</priority>
</rule>
</rules>
</profile>
```



## 8 Software performance testing

One major problem is the complexity of the Cortex framework as growing complexity often implies performance flaws. To define and measure the shifts of data between the different layers of the software and the corresponding effects on the performance of the system, extensive load tests have to be implemented.

Normally, performance testing tools request a resource of the system to be examined and measure the time for the arrival of a response. Ideally this is done in a distributed manor, as of today virtually no software runs singlethreadedly.

### 8.1 Goals

The goals of performance testing do not primarily span finding errors but to measure scalability. A system, that behaves and scales very good in normal debugging environments during the mandatory unit tests in the course of implementation, might behave utterly though when under heavy load from thousands of requests in real-world environments.

To measure this scalability, software performance tests are implemented, that simulate this real-world behaviour in order to define boundaries, which can later be declared as hard rules for the mode of operation.

The main question though, standing all behind the concept of load testing would be:

*"how much do i have to spend for hardware to deal with **this** amount of users?"*

### 8.2 Definitions

Most of the implemented tests cover Cortex's various REST-endpoints (see REST Endpoints [2] for more information) as those form the basis of all logic to be presented to the user and cover almost every relevant internal function and interface, that can be load-tested.

For the sake of completeness, some vocabularies are defined below, that are referenced throughout the rest of this chapter consistently.

#### 8.2.1 Load tests

Load tests form the basis of performance testing software, as they simulate real-world situations, in which multiple users request different functions and data at random. This differs from unit

tests, which do usually test one discrete function at a time. The main goal of this test is to find unexpected behaviour, like performance flaws of the database when reading and writing simultaneously with multiple threads or - even worse - deadlocks, that under clean testing environments are unlikely to be found.

At the same time, load tests represent the most complex manifestation of a performance test, as the randomizers for the different request types have to permutate over every possible value and interface they know - which can be quite a lot in our case.

### 8.2.2 Performance tests

Performance tests repeat single operations with an intentionally average workload. This is important, to measure

1. the scalability of the specific functions discretely
2. possible increases of system resources or load due to ie. caching/cleanup issues over time

### 8.2.3 Stress tests

The main purpose of stress tests is to increase the load on the tested functions up to a previously defined maximum level, with the goal to measure the scalability of the software with increasing load levels. An ideal result would be a no-difference in the response times, but this might be a slightly too optimistic value.

So we define the real-world-optimum as a linear scaling of the response time in correspondence to increasing numbers of threads and requests per minute. Later on this value can be used to specify the hardware requirements with a simple  $f(users) : cost$ -function.

Quite interesting might also be to measure the correspondence of the size of the requested or returned data payload compared to a growing number of requests as this might as well pose a bottleneck.

### 8.2.4 Failover tests

Failover tests are strongly related to Stress tests [8.2.3] in matters of increasing the number of requests over time. However the main, grave difference is, that there is no upper boundary. Some might depict this as a DDOS-attack - which in fact has the same goal: wrecking the system.

There's one major difference, though: Failover tests measure the time, how long the server works flawlessly free of errors and responds in an acceptable manor.

In our case, this might include, whether the Cortex in a distributed setup still reads and writes data as intended when under heavy load, as distributedness implies asynchronously share data on all involved servers.

## 8.3 JMeter

The major tool for performance tests used internally in the DDB project is Jakarta JMeter, which can be downloaded under the APL v2 License from the following URL:

<http://jakarta.apache.org/jmeter/>

JMeter offers various possibilities for performance testing software, some of which are:

**GUI** for easier and more intuitive setup of tests - this kills two birds with one stone, because the overhead of understanding and explaining the function of tests for 3rd-person testers is held at a reasonable level, too.

**Simulating Users** via threads, so that parallel requests can be simulated. Each of the threads can be set up to behave autarchically, so has it's own discrete testdata or random counter.

**Distributing** tests to different machines and controlling them from one host. As testing itself produces some load, too, it can distort the testing results just by generating multiple tests.

**Controlling** scripts, which can simulate some default behaviour like running  $n$  threads "at once" on a single endpoint to simulate a short maximum peak load and measure the return-to-normal-state time of the server.

...

## 8.4 Test stories

Definitions of test stories help both technicians and non-technicians understand the ideas behind the tests. To simulate better user behaviour, timings between the test requests are also provided as an example. These timings are not hardcoded and can vary depending on many factors like specific user behaviour. So far we have identified and defined the following test categories:

### 8.4.1 synthetic tests

As there will be heavy load on these basic components, the following tests implement a sharp spot. Timings are not provided here as the main focus is not to replicate user behaviour but to extend tests to heavy load on specific components.

#### preview test

JMeter takes number of IDs from the DDB and requests only the previews. As all previews are also stored in the SolR backend, the previews do not need to be requested through the file-based backend.

story	action
request a number of IDs	/search/facets?offset&rows
read previews list	read search results

### view test

JMeter takes a number of IDs from the DDB and requests only the views.

story	action
request a number of IDs	/search/facets?offset&rows
read multiple items	/access/identifer/view

### keyword test

a simulated user takes a number of IDs from the DDB and extracts random search keywords from their views. These keywords are then used for subsequent tests.

story	action
perform a paged wildcard search	/search/facets?offset&rows
request a random view	/access/identifiers/view
extract a search keyword	regex extraction

## 8.4.2 simulations

Simulating users is the second category of tests. We provide example timings for every real request and simulated user behaviour but these can vary due to many simulated factors like specific user training up to client-, and hardware speed.

### typical search

A simulated user searches and views the results

story	action	seconds
enter random search keyword	/search/facets	5 - 10
read search results	wait	3-10
paging	/search/facets?offset	3-10
read item	/access/identifer/view	7

### facet test

a simulated user searches and filters the results through facets

story	action	seconds
enter search keyword	/search/facets	5 - 10
read search results	wait	3-10
facetting search	POST /search/facets	3-10
read search results	wait	3-10
read item	/access/identifer/view	7

### weighted component test

a simulated user takes a number of IDs from the DDB and requests a random but weighted single component. The weights have to be previously defined.

story	action	seconds
enter search keyword	/search/facets	5 - 10
read search results	wait	3-10
paging	/search/facets?offset	3-10
read item	/access/identifer/view	7

### 8.4.3 journeys

TODO:

### 8.4.4 replicating real users

TODO:

## 8.5 Test implementations

This section describes the premises of the tests and some of the implemented logic.

### 8.5.1 Problems and solutions

#### Caching

Because of the architecture of the framework, simply asking the rest interfaces for one value might result in distorted test results, because some of the layers, like webserver have request-response caching technologies. Of course, this is a feature, that is of great use in the internet to minimize workload of the servers, but in our case we do not want to test the caching capabilities of webserver or proxies, but rather the speed of the application logic behind our software.

To circumvent this shelf, we utilize a randomizing algorithm in the testclients, that tries to lever out the caching-mechanisms simply stoically generating new requests.

## Threads, concurrency and timing

TODO:

### Used data

So far, there is no REST endpoint announcing a list of identifiers in the Cortex. As this is needed by our randomized test setup, we have to extract these IDs from the repository by hand (okay, let's say by script). This data is then transformed to a simple `.csv` format, which can easily be understood by JMeter. The same applies to ingest-events.

## 8.5.2 Concrete test implementations

TODO:

The tests are still in development, so is this document.

### REST: Access Tests

The REST access endpoint will be - alongside the search endpoint maybe - the most heavily used endpoint in the Cortex. So this has to be the most strictly tested interface.

Tests in this category so far span all methods mentioned in Definitions [8.2].

### REST: Reports Tests

### REST: Ingest Tests

### REST: User Profile Tests

## 8.5.3 Combined tests

## 8.6 Results

So far, not all tests have yet been implemented, nor run distributedly. This is on schedule, though. Anyway, first results and simple tests show the scaling of the REST-endpoints Data Access [2.1] and Search and Facets [2.3] seems to be of a linear characteristic. The Reporting Interface [2.5] endpoint refuses operation with normal harddrives and too much data, as it uses too much resources to generate the reports.

## 9 Integration Tests

### 9.1 Introduction

Unlike Unit Tests, which only focus on single parts of the implementation, the purpose of Integration Tests is to test a software-system on the whole. They execute all components and cover all features that are implemented. The goal of these tests is to ensure, that a certain release that is being deployed is stable. Side-effects, that only occur, when different components interact with each other can only be covered by these kind of tests.

### 9.2 Full-Cortex-Test

One part of the Cortex-Test-Plan is the Full-Cortex-Test (following FCT), which is an implementation of an Integration Test for Cortex. It is located in the project CS\_Integration Test, the package is `de.fhg.iais.cortex`.

#### 9.2.1 Architecture

Although the FCT is not a Unit-Test by definition, it is implemented as a JUnit-Test. Reasons for that are on the one hand, the architecture of Maven, which can execute JUnit-Tests automatically, on the other hand the assertion feature of JUnit, including its Eclipse-integrated GUI for analyzing test-results.

Since an Integration Test should simulate the behavior of the software during its normal operation, it is necessary to run it with different configurations. For that reason, the FCT can be configured on two different sides:

- Run configurations
- Test data sets

The folder `CS_IntegrationTest/conf` contains eight properties files. The property being used can be defined by the local constant `TESTPROPERTIES` in the FCT. Depending on which file is being loaded, the test can be executed with `direct-ingest` or `MQ-ingest` option, including or excluding binary files, and running in a functional testing mode or performance-testing mode. By edition the constant `MIXFILENAME` you can choose, which inbox should be loaded and used for testing. By defining an inbox you can define which kind of data is being tested. A

certain behaviour, which is expected for a certain inbox can be defined by a properties file which is related to the inbox. All inboxes are stored in `Cortex/conf/testdata/inboxes`.

## **9.2.2 Design and application flow**

The FCT can be divided into three parts:

- Cortex ETL and Ingest operations
- Performance testing
- Functional testing

The ETL and Ingest operations are the basis for the cortex-platform. They are fundamental for deploying data. The Test starts the process in its order, which is predefined by the system:

- An ASC is being started. The ASC takes an inbox with metadata (and binaries) and transforms them.
- The ASC starts the ingest process. Cortex is now being filled with data

This process is only being launched by the FCT. All operations are being executed in a black-box from the tests point of view.

After the ingest, the testing can start. By default, the FCT is a functional test. When the ingest-process is finished, it is validated and the features of the Cortex-Platform are being tested. Alternatively, the FCT can be executed as performance test. In this case, the test ends after the ingest process and gives a print-out of all recorded chronometric information, which are the duration of the ASC processes, the ingest-process and the overall duration.

## **9.3 Functional tests**

### **9.3.1 Ingest validation**

The ingest process is validated by the ingest-reports. Reports contain information about the ingest status (success / error) of an item and additional information that are necessary in case of errors. When the ingest-process is finished, each report is being checked for its success state. If there is an unexpected error (you can define expected errors before launching the test) the test fails immediately.

### **9.3.2 Rest-Interface-Test**

The Rest-Interface-Test performs REST calls on the interfaces provided by Cortex. It is launched by the FCT. The FCT provides the necessary test-data for the REST-test by picking out one

random item per ingest-id and passing item-id as well as the data-path to the rest-test. The REST-test grabs the expected data from the filesystem by using the provided data-path, and compares those values to the results provided by the rest-call.

Currently, the REST-test covers almost all access-calls except for the access-methods of binary files (since they are not finally implemented).

## **9.4 Known limitations / Problems / TODOs**

### **9.4.1 MQ-Ingest finish notification**

The FCT depends on the assumption, that we know when the ingest is finished. During Direct-Ingest Mode, that is being done by a .lock-File. As long as it exists in the outbox, the ingest is running. In MQ-Ingest mode, this validation is not possible yet.

Currently the test just waits for a console-input by the user (press enter when ingest is finished) in MQ-mode.

### **9.4.2 Report-handling**

The current implementation of the report validation is not very efficient. It stores all reports first and checks them afterwards. That does not work with a huge amount of data (java heap out of memory), so the functional test fails. Since reports are currently being restructured (which will solve the problem immediately), this part should be changed when reports are finished.

### **9.4.3 Starter**

Since there is a bug in SolR, which prevents SolR from starting out of a Maven-test scope, we can't start the Cortex Server on the fly during the test. Starting Cortex currently has to be done by hand before launching the test. In order to be sure, that it is running the FCT tries to open a server socket using port 8080, which fails if Cortex is already running. If the server socket connection is successful the test fails, since Cortex is not running. Without that mechanism the test could run into an endless loop since it waits for an ingest-end which will never occur (because ingest never started).