# D5.3: OKKAM API Reference Guide

<table>
<thead>
<tr>
<th>Document Number</th>
<th>D5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Title</td>
<td>OKKAM API Reference Guide</td>
</tr>
<tr>
<td>Version</td>
<td>1</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Work Package</td>
<td>WP5</td>
</tr>
<tr>
<td>Deliverable Type</td>
<td>Report</td>
</tr>
<tr>
<td>Contractual Date of Delivery</td>
<td>30/04/2009</td>
</tr>
<tr>
<td>Actual Date of Delivery</td>
<td></td>
</tr>
<tr>
<td>Responsible Unit</td>
<td></td>
</tr>
<tr>
<td>Contributors</td>
<td>EPFL, MAC, Trento, L3S, DERI, Malaga, DERI, ExpertSystem</td>
</tr>
<tr>
<td>Keyword List</td>
<td></td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU</td>
</tr>
</tbody>
</table>
## Change History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Status</th>
<th>Author (Company)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17/03/2009</td>
<td>Draft</td>
<td>Zoltan Miklos (EPFL)</td>
<td>First draft, document skeleton</td>
</tr>
<tr>
<td>2</td>
<td>19/03/2009</td>
<td>Draft</td>
<td>Zoltan Miklos (EPFL)</td>
<td>Initial comments incorporated</td>
</tr>
<tr>
<td>3</td>
<td>14/04/2009</td>
<td>Draft</td>
<td>Daniele Cordioli (ExpertSystem)</td>
<td>Input ExpertSystem</td>
</tr>
<tr>
<td>4</td>
<td>14/04/2009</td>
<td>Draft</td>
<td>Joe Cantwell, John O’Flaherty (MAC)</td>
<td>Input MAC</td>
</tr>
<tr>
<td>5</td>
<td>16/04/2009</td>
<td>Draft</td>
<td>Katerina Ioannou, Claudia Niederée (L3S)</td>
<td>Input L3S</td>
</tr>
<tr>
<td>6</td>
<td>16/04/2009</td>
<td>Draft</td>
<td>Giovanni Tumarello, Robert Fuller (DERI)</td>
<td>Input DERI</td>
</tr>
<tr>
<td>7</td>
<td>20/04/2009</td>
<td>Draft</td>
<td>Zoltan Miklos (EPFL)</td>
<td>Input EPFL, further editing</td>
</tr>
<tr>
<td>8</td>
<td>21/04/2009</td>
<td>Draft</td>
<td>Hristo Koshutanski (UMA), Antonio Maña (UMA)</td>
<td>Security APIs description added.</td>
</tr>
<tr>
<td>9</td>
<td>22/04/2009</td>
<td>Draft</td>
<td>Themis Palpanas (Trento)</td>
<td>Lifecycle API description added</td>
</tr>
<tr>
<td>10</td>
<td>5/5/2009</td>
<td>Draft</td>
<td>Giovanni Tumarello</td>
<td>Peer review by the deliverable mentor</td>
</tr>
<tr>
<td>12</td>
<td>26/5/2009</td>
<td>Draft</td>
<td>Zoltan Miklos</td>
<td>Incorporating further comments by Robert Fuller, Giovanni Tumarello (DERI), Paolo Bouquet (UNITN)</td>
</tr>
<tr>
<td>13</td>
<td>27/05/2009</td>
<td>Final</td>
<td>Zoltan Miklos</td>
<td></td>
</tr>
</tbody>
</table>
Executive Summary

In this document we provide a guide to the OKKAM internal and external API.

The first part of this document describes the public OKKAM API. Additionally to the API description it contains some example code for application developers or ENS users. The text complements and extends the information available on http://www.okkam.org/apis.

The second part focuses on the internal APIs. An organic part of this deliverable is the javadoc documentation of the APIs, which is available at http://gforge.okkamdev.org/TechnicalDocs/index.html. This document concentrates on providing a guide and an overview of the APIs, while the actual technical details are described more precisely and more in detail in the javadoc documentation.

The deliverable also contains information on the code commenting and documenting directives and some other software code related quality assurance measures, agreed by OKKAM partners.
Table of Contents

1. INTRODUCTION ...................................................................................................................................................... 6
   1.1. WHAT IS OKKAM? ............................................................................................................................................... 6
   1.2. THE SCOPE OF THE API GUIDE ....................................................................................................................... 6
   1.3. QUALITY ASSURANCE ....................................................................................................................................... 7
   1.4. DOCUMENT STRUCTURE ................................................................................................................................. 8

2. PUBLIC OKKAM API .................................................................................................................................................... 9
   2.1. ENS WEB SERVICES API ................................................................................................................................... 9
       2.1.1. method "startSession" ............................................................................................................................... 9
       2.1.2. A MessageResult ...................................................................................................................................... 10
       2.1.3. Method "startProcess" ............................................................................................................................. 10
       2.1.4. Search an entity: method “findEntity” ....................................................................................................... 11
       2.1.5. A MatchingCandidate ............................................................................................................................. 12
       2.1.6. Get created entity and status information: method “getSelectedEntity” .................................................... 12
   2.2. APPLICATION SERVICES API .......................................................................................................................... 13
       2.2.1. Named Entities extraction .......................................................................................................................... 13
       2.2.2. Named entities with Okkam Entities (from plain text) ................................................................................. 13
       2.2.3. Named entities with Okkam Entities (from URL) ....................................................................................... 13
       2.2.4. Okkam Queries ........................................................................................................................................... 14
       2.2.5. RDFa enrichment ......................................................................................................................................... 14
       2.2.6. OKKAMize Web Source ........................................................................................................................... 14
   2.3. BULK ENTITY FINDER API .............................................................................................................................. 16
       2.3.1. Input File .................................................................................................................................................... 16
       2.3.2. Output File ............................................................................................................................................... 16
       2.3.3. The service ................................................................................................................................................ 16

3. INTERNAL OKKAM API ............................................................................................................................................ 18
   3.1. OKKAM CORE API .............................................................................................................................................. 18
   3.2. STORAGE API ...................................................................................................................................................... 19
       3.2.1. Functionality ............................................................................................................................................... 19
       3.2.2. Example .................................................................................................................................................... 20
   3.3. MATCHING API .................................................................................................................................................. 22
       3.3.1. Requesting Entities .................................................................................................................................... 22
       3.3.2. Result format ............................................................................................................................................. 24
   3.4. ENTITY LIFECYCLE API ....................................................................................................................................... 25
       3.4.1. Creating a New Entity .................................................................................................................................. 25
       3.4.2. Update Entity ............................................................................................................................................. 27
       3.4.3. Auxiliary methods .................................................................................................................................... 27
   3.5. SECURITY API ...................................................................................................................................................... 28
       3.5.1. Overview of authentication and access control .......................................................................................... 28
       3.5.2. Functionalities of the proxies .................................................................................................................. 29
       3.5.3. Interaction with the proxies ..................................................................................................................... 29

4. CONCLUSIONS AND FUTURE PLANS .................................................................................................................. 32

5. REFERENCES ............................................................................................................................................................... 33

ANNEX A: JAVADOC .................................................................................................................................................. 34
   A.1 HOW TO WRITE DOC COMMENTS FOR THE JAVADOC TOOL ......................................................................... 34
       A.1.1 Who Owns and Edits the Doc Comments ................................................................................................. 35
   A.2 REQUIREMENTS FOR WRITING OKKAM API SPECIFICATIONS ................................................................. 35
       A.2.1 Top-Level Specification .................................................................................................................................. 36
       A.2.2 Component Specification ............................................................................................................................. 36
       A.2.3 Method Specification ..................................................................................................................................... 36
## Glossary

<table>
<thead>
<tr>
<th>OKKAM</th>
<th>IST 7th Framework Research Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENS</td>
<td>Entity Name System (developed by the OKKAM project)</td>
</tr>
<tr>
<td>EPFL</td>
<td>Ecole Polytechnique Fédérale de Lausanne, Swiss Federal Institute of Technology</td>
</tr>
<tr>
<td>UNITN</td>
<td>University of Trento (Italy)</td>
</tr>
<tr>
<td>MAC</td>
<td>National Microelectronic Application Center (Ireland)</td>
</tr>
<tr>
<td>DERI</td>
<td>Digital Enterprise Research Institute (Ireland)</td>
</tr>
<tr>
<td>ECSSE</td>
<td>Entity Centric Semantic Search Engine</td>
</tr>
<tr>
<td>UMA</td>
<td>University of Malaga (Spain)</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFa</td>
<td>Resource Description Framework in attributes</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>UTF-8</td>
<td>Unicode Transformation Format (8 bit)</td>
</tr>
<tr>
<td>Sindice</td>
<td>Sindice is a research project at DERI</td>
</tr>
</tbody>
</table>
1. Introduction

1.1. What is OKKAM?
OKKAM is a Large-Scale Integrating Project funded by the European Commission under the 7th Framework Program (FP7) until June 2010.

The OKKAM project aims at enabling the Web of Entities, namely a virtual space where any collection of data and information about any type of entities (e.g. people, locations, organizations, events, products, ...) published on the Web can be integrated into a single virtual, decentralized, open knowledge base (like the Web did for hypertexts.)

OKKAM will contribute to this vision by supporting the convergence towards the use of a single and globally unique identifier for any entity which is named on the Web. The intuition of the project is that the concrete realization of the Web of Entities requires that we enable tools and practices for cutting to the root the proliferation of unnecessary new identifiers for naming the entities which already have a public identifier (the OKKAM's razor). Therefore, OKKAM will make available to content creators, editors and developers a global infrastructure and a collection of new tools and plugins which support them to easily find public identifiers for the entities named in their contents/services, use them for creating annotations, build new network-based services which make essential use of these identifiers in an open environment (like the Web or large Intranets).

To realize this vision, OKKAM proposes the following roadmap:

- Providing a scalable and sustainable infrastructure, called the Entity Name System (ENS), for making the systematic reuse of global and unique entity identifiers not only possible, but easy and straightforward. The ENS will be a distributed service which permanently stores identifiers for entities and provides a collection of core services (e.g. entity matching, ID mapping and resolution) needed to support their pervasive reuse;
- bootstrapping and enabling the fast growth of Web of Entities by fostering the creation of OKKAMized content (i.e. content where entities are named or annotated with OKKAM IDs) in OKKAM-empowered applications (i.e. applications which can interact with the ENS for getting and reusing identifiers);
- showcasing the benefits of enabling the Web of Entities and, more in general, of an entity-oriented approach to content and knowledge management by building relevant applications on top of the new infrastructure in three important areas: information retrieval and semantic search, content authoring (more specifically, in scientific publishing and news production) and organizational knowledge management.

More up-to-date information can be found on the OKKAM webpage: http://www.okkam.org.

1.2. The scope of the API guide
This deliverable gives an overview of the OKKAM ENS APIs. First we describe the APIs which are available for applications to consume the services of the ENS, and then we describe the internal APIs. The description of the external API is intended to help developers writing OKKAM related applications, for example retrieving unique identifiers or extracting entities from a text source. The document should foster the early adoption of OKKAM, however a more detailed developers guide
shall be prepared later (Deliverable D5.7). The internal API description on the other hand should help OKKAM developers to orient in the javadoc documentation.

The current version of software foresees a Web-service style communication for accessing the ENS functionality. The Application Services API, which enables to extract entities from documents or Web-resources, also foresees communication via Web Services. Examples and tutorials how to use the services is available online for our users. We also included a chapter in this deliverable on Sindice, in particular the description of APIs related to OKKAM for realizing an ECSSE.

The internal APIs include the storage-, the matching-, the lifecycle, the security and the OKKAM core APIs. The document is not intended to be the documentation of the APIs, rather a guide, which helps the users to orient.

We discuss the quality assurance issues related to code documentation in the following section.

### 1.3. Quality assurance

OKKAM deliverable D5.1\(^2\) describes the software development guidelines, quality assurance procedures and generic commenting practices to be used in the development of the OKKAM system using the Development Management Platform at [http://gforge.okkamdev.org](http://gforge.okkamdev.org). To ensure the quality and utility of the technical documentation generated by the Javadoc tool, the following additional requirements are required of OKKAM Development Partners:

A. It is recommended that Developers use the Sun conventions for writing documentation comments as given at the Sun “How to Write Doc Comments for the Javadoc Tool” website.\(^3\) While OKKAM API Specifications should be documented as described at the Sun “Requirements for Writing Java API Specifications”\(^4\), which covers requirements for packages, classes, interfaces, fields and methods to satisfy testable assertions. See Annex A for an introduction and overview.

B. Partners can code their components as normal, taking care to comment each method etc. with Javadoc compatible syntax and useful comments (as in A).

C. A Technical Documentation Umbrella Project will be available on [http://gforge.okkamdev.org](http://gforge.okkamdev.org) using Subversion:Externals which links each component to this project. Javadoc will be automatically run against this Umbrella Project using Ant\(^5\) after each minor release of the OKKAM ENS system, and will compile the documentation for each component referenced in the project.

---

1 In later versions Representational State Transfer (REST)-style interaction might be added, to complement the web-service interaction.


So, except for the adoption of the best practice documentation commenting in step A, the Javadoc tool places no further requirements on Developers, and provides visibility to the latest available technical documentation at http://gforge.okkamdev.org/TechnicalDocs/index.html.

From the beginning all OKKAM Components are included in this automatic process, so it is expected that while the initial Technical Documentation leading up to ENS V2.0 at http://gforge.okkamdev.org/TechnicalDocs/index.html will not be as complete as desired, it will improve over time and iterations of the system, as Developers adopt the documentation commenting standards specified in step A above. It is planned that by ENS V3 this process will be the norm within the OKKAM Development Community.

1.4. Document Structure

The document structured as follows:

In Chapter 2, we present the public OKKAM APIs, and the related APIs of Sindice ECSSE. Chapter 3, gives an overview of the internal OKKAM APIs, in particular, Storage-, Matching-, Entity Lifecycle- Security, and OKKAM core APIs. The annex contains basic guidelines for commenting and documenting the code developed by OKKAM partners. An organic part of this deliverable is the javadoc documentation of the internal APIs, which is available at http://gforge.okkamdev.org/TechnicalDocs/index.html.
2. Public Okkam API

The ENS stores unique identifiers and offers services to applications to retrieve them. The ENS services are available via Web Services. This API is described in the following sections, together with some reference to the examples. The Application Services API offers a service to automatically extract entities from documents and Web resources. These services are described in the following sections. We also describe the OKKAM relevant APIs of the Sindice ECSSE. 6

Examples and tutorials for creating entities both independently from programming language and for java programmers are available at http://www.okkam.org/apis/tutorials/create-entity.

2.1. ENS Web Services API

A description of the Web Services API is publicly available at http://www.okkam.org/apis/web-service-api, which contains further examples as well.

WSDL URL: http://api.okkam.org/okkam-core/services/WebServices?wsdl

SERVICE ADDRESS: http://api.okkam.org/okkam-core/services/WebServices

In the following we describe the available functionality.

2.1.1. Method "startSession"

To run Okkam Web Services one has to pass a SessionMetadata object. To get one, one has to use the method startSession as described below.

```xml
<element name="startSession">
  <complexType>
    <sequence>
      <element name="authKey" type="xsd:string" />
    </sequence>
  </complexType>
</element>
```

Input:

- `authKey` here you pass the authKey "asdfghjkl" (second line of your keyboard :D) in a later version of Okkam Web Services, you will need to register to obtain your personal authKey

As a response the application receives a `StartSessionResultClient` object:

```xml
<complexType name="StartSessionResultClient">
  <sequence>
    <element name="messageResult" nillable="true" type="tns2:MessageResult" />
    <element name="sessionMetadata" nillable="true" type="tns1:SessionMetadata" />
  </sequence>
</complexType>
```

6 We decided to include some details about Sindice APIs in this document, because they are already implemented and functional and they might play an important role to populate the OKKAM repository.
Result:

- `resultMessage` represents status of the result (request ok, errors appeared, illegal authorization, etc.) - **NOTE**: all Okkam Web Service methods a ResultMessage object with this kind of information - see a detailed definition of it below.
- `sessionMetadata` should be stored locally, as one will need it to call other Okkam Web Services, for example "findEntity".

Please ignore all "nillable=true" statements, they are due to the prototype status of the Web Services.

### 2.1.2. A MessageResult

```xml
<complexType name="MessageResult">
  <sequence>
    <element name="code" type="xsd:int" />
    <element name="message" nillable="true" type="xsd:string" />
    <element name="successful" type="xsd:boolean" />
  </sequence>
</complexType>
```

Elements: `code`:

- `message`: a human readable string, describing the code
- `successful` is a flag: it is true if everything is ok (if an OK code was not sufficient)

### 2.1.3. Method "startProcess"

Each session can contain processes. A process could be an entity creation or update (supposed to be supported after Okkam v1.1).

```xml
<element name="startProcess">
  <complexType>
    <sequence>
      <element name="metadata" type="tns1:SessionMetadata" />
      <element name="authKey" type="xsd:string" />
    </sequence>
  </complexType>
</element>
```

Input:

- `metadata` obtained from a call to startSession
- `authKey` again "asdfghjkl", in future versions of Okkam Web Services, different Web Service methods will require different access rights and therefore maybe different authKeys for different methods

This call will return a `StartProcessResultClient` object defined below.
Output:

- `messageResult` as described before
- `sessionMetadata` as described before, but now containing a valid process ID, too.

2.1.4. Searching for an entity: the method "findEntity"

The method returns a list of entities that match the given query.

Input:

- `query` a string with your query. It can either contain keywords, e.g. "Paolo Bouquet", or a structured query like "QUERY{firstname=paolo}METADATA{entityType=person}
- `metadata` a SessionMetadata as mentioned before
- `authKey`: again "asdgfhjkl"

This call will return a `FindResultClient` object defined below.

Output:

- `matchingCandidate` is a list of matching entities and their relevance/closeness - see below
- `messageResult` as described before
- `sessionMetadata` as described before
2.1.5. The type MatchingCandidate

```xml
<complexType name="MatchingCandidate">
  <sequence>
    <element name="XML" nillable="true" type="xsd:string" />
    <element name="oid" nillable="true" type="xsd:string" />
    <element name="sim" type="xsd:double" />
  </sequence>
</complexType>
```

Elements:
- **XML**: a string containing URL-encoded XML that represents an entity (see Entity Schema). In java we use JAXB to convert an Entity to XML and backwards
- **oid**: the Okkam ID, aka ENS ID or ENS Identifier, of the entity given in URL-encoded XML above
- **sim**: similar measure, closeness, etc. A number <1 that describe the quality of a result

2.1.6. Get created entity and status information: method "getSelectedEntity"

The method returns a status message and, if the process has been finished successfully, an entity.

```xml
<element name="getSelectedEntity">
  <complexType>
    <sequence>
      <element name="metadata" type="tns1:SessionMetadata" />
      <element name="authKey" type="xsd:string" />
    </sequence>
  </complexType>
</element>
```

Input:
- **metadata** a SessionMetadata as mentioned before
- **authKey**: again "asdgfhjkl"

This call will return a FindResultClient object defined below.

```xml
<complexType name="FindResultClient">
  <sequence>
    <element name="entity" nillable="true" type="xsd:string"/>
    <element name="messageResult" nillable="true" type="tns2:MessageResult" />
    <element name="sessionMetadata" nillable="true" type="tns1:SessionMetadata" />
  </sequence>
</complexType>
```

Output:
- **entity** the created entity as URL-encoded XML or null, if the process is still in progress or has been aborted
- **messageResult** as described before
- **sessionMetadata** as described before
2.2. Application Services API

The Application Services API allows that applications automatically extract the following elements from a text document:

- the named entities like People, Companies, Location, and other
- the OKKAM entities
- the OKKAM core queries generated with the extraction of the named entities
- the RDFa enrichment
- Asynch process of urls to generate new OKKAM entities

**WSDL URL:** [http://host.expertsytem.it/OkkamWebService/services/OkkamWebService?wsdl](http://host.expertsytem.it/OkkamWebService/services/OkkamWebService?wsdl)

2.2.1. Named Entities extraction

The API call is as follows:

```java
String highlightTextWithOkkamQueries(String activationKey, int language, String plainText)
```

where:

- `activationKey` is the API access key;
- `language` 0=Italian, 1=English
- `plainText` represent the content to be analyzed

The result is in XML format and contains the extracted named entities and the queries that allow you to execute OkkamCore calls.

2.2.2. Named entities with Okkam Entities (from plain text)

The API call is as follows:

```java
String highlightTextWithOkkamResults(String activationKey, int language, String plainText)
```

where:

- `activationKey` is the API access key;
- `language` 0=Italian, 1=English
- `plainText` represent the content to be analyzed

The result is in XML format and contains the extracted named entities, the queries that allow you to execute OkkamCore calls and the result of the OkkamCore calls.

2.2.3. Named entities with Okkam Entities (from URL)

The API call is as follows:

```java
String okkamizeUrl (String activationKey, int language, String url)
```

OKKAM - 215032 Version 13.0 Page 13 of 37
where:

- `activationKey` is the API access key;
- `language` 0=Italian, 1=English
- `url` represent the url to be analyzed

The result is in XML format and contains the extracted named entities, the queries that allow you to execute OkkamCore calls and the result of the OkkamCore calls.

### 2.2.4. Okkam Queries

The API call is as follows:

```java
String okkamQueries (String activationKey, int language, String plainText)
```

where:

- `activationKey` is the API access key;
- `language` 0=Italian, 1=English
- `plainText` represent the content to be analyzed

The result is in XML format and contains the queries that allow you to execute OkkamCore calls.

### 2.2.5. RDFa enrichment

The API call is as follows:

```java
String microformatTextEnricher (String activationKey, int language, String plainText)
```

where:

- `activationKey` is the API access key;
- `language` 0=Italian, 1=English
- `plainText` represent the content to be analyzed

The result is the plain text converted in RDFa standard.

### 2.2.6. OKKAMize Web Source

**Start process**

The API call is as follows:

```java
String okkamizeWebSource (String activationKey, int language, String url, int depth, String[] inclusion, String[] exclusion)
```

where:

- `activationKey` is the API access key;
- `language` 0=Italian, 1=English
• *url* is the crawler start point
• *depth* of the crawler
• *inclusion* represents a list of the included URLs
• *exclusion* represents a list of the excluded URLs

The result is in XML format and contains the operation identifier, the operation status and a message. Below is an example of a possible response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<okkamResponse operationId="c30e4991ae434ee39d0294a08f5889e5" status="running">
<message><![CDATA[operation started]]></message>
</okkamResponse>
```

**Operation status**

The function to determine the operation status is

```java
String getOperationStatus (String activationKey, String operationId)
```

where:

- *activationKey* is the API access key;
- *operationId* is the identifier

The result is in XML format and it contains the operation identifier and the operation. Below is an example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<okkamResponse operationId="c30e4991ae434ee39d0294a08f5889e5" status="running" />
```

**Operation result**

The operation result can be obtained as follows:

```java
String getOperationResult (String activationKey,String operationId)
```

where:

- *activationKey* is the API access key;
- *operationId* is the identifier

The result is in XML format and it contains the local machine path where the extracted entities were saved in a defined format that can be inserted into the Okkam Repository.
2.3. Bulk Entity Finder API

The bulk finder service allows uploading a file containing many findEntity queries to an ftp server, and retrieving the result from the server when all the queries have been processed.

The procedure to use the service is the following:

- One has to upload a file containing queries to an ftp server,
- send the ftpUrl (with credentials if required) to the submit service
- Use the status service with ftpUrl to check progress
- Retrieve the processed file from the ftp server
- If required use the cancel service to cancel processing of a file.

The service is available at http://okkam.sindice.com/bulk/finder

For using each of the services, the ftpUrl parameter is required.

2.3.1. Input File

The Input file is a text file, which should contain one query per line. One has to use UTF-8 file encoding if there are any special characters contained. A sample input file is available at http://okkam.sindice.com/bulk/sample_input.txt

2.3.2. Output File

The output file will be placed onto the ftp server next to the input file with the file extension '.result'. During processing a file with the name '.result.part' may be present on the ftp server. An example can be found at http://okkam.sindice.com/bulk/sample_results.xml. An other example is available at http://okkam.sindice.com/bulk/sample_results_without_candidates.xml, which is generated by including parameter omitCandidateXml=true in request.

2.3.3. The service

The restful service can be consumed using two different formats: the bulk entity finding service can produce html output, which is suitable for human interaction, and also xml output, which is more suitable for automated processing for applications. The three available operations are submit a request, check the status of an operation, cancel the processing.

The server can be configured with passwords for accessing ftp servers if required. Otherwise, one has to upload a file to ftp server and includeing the ftp credentials in the url.

For each of the operations, the parameter ftpUrl=ftp://<username:password>@host:port/path/to/query.txt is required. (username:password credentials are omitted if these have been already configured on the server)

2.3.3.1 Submit

The process can be started as follows:

```html
```

```xml
```
NB: To omitCandidateXml from the result, include the following parameter in the http request: omitCandidateXml=true

### 2.3.3.2 Status

One can obtain the status of the process as follows:

<table>
<thead>
<tr>
<th>Format</th>
<th>URL</th>
</tr>
</thead>
</table>

### 2.3.3.3 Cancel

If necessary, the bulk entity finding process can be interrupted as follows:

<table>
<thead>
<tr>
<th>Format</th>
<th>URL</th>
</tr>
</thead>
</table>
3. Internal OKKAM API

The internal architecture of a single ENS node consists of the following components:

- **Core**: the integrating component that provides interactions between the components (managed by WP5);
- **Matching**: responsible for entity matching (managed by WP3);
- **Access (security)**: component responsible for security-related functionality, e.g. access control (managed by UMA, MAC and UNITN in the context of PCA-5 and WP5);
- **Life-cycle**: responsible for everything that concerns the evolution of data in the ENS (managed by WP6);
- **Storage**: responsible for storage of data in the ENS (managed by WP2).

In this chapter we give an overview of the internal OKKAM APIs. This overview should help OKKAM developers to orient in the javadoc documentation, which is available at [http://gforge.okkamdev.org/TechnicalDocs/index.html](http://gforge.okkamdev.org/TechnicalDocs/index.html), and is an organic part of this deliverable.

### 3.1. Okkam core API

The internal architecture of a single ENS node consists of the following components: OKKAM core, matching, lifecycle, access and storage. The components are connected via Java interfaces which are defined in the Core. This guarantees compatibility of a component with the overall architecture at compilation level and prevents ad-hoc interface changes by component providers.
The core performs runtime loading of the individual components, so that – in theory – these can be replaced seamlessly without recompilation or code changes of the complete system.

If a component needs to communicate with another one, it requests a channel to this component from the Core. The Core is furthermore responsible for managing sessions that require stateful processes which are exposed over a number of different web services (e.g. for entity creation).

OKKAM core and its components are bound together as a Web Application Archive (WAR) and deployed in an instance of the high-performance application server JBOSS.

The main role of the OKKAM core API is to realize the integration of the individual components.

3.2. Storage API

This section describes the Storage API. The main functionalities of the storage layer are to provide means to insert and to delete entities or to update an entity in the storage and also to enable efficient the querying of the entity store. In the future versions, the storage API will also provide functions which return some statistical information about the stored entities and also procedures which are specifically needed by the Lifecycle component.

3.2.1. Functionality

3.2.1.1 Insertion of an Entity

The current entity representation defines an entity in an XML format. In AnormStore entity is both stored as well as indexed. AnormStore writes the contents of an entity as a blob string identified by Okkam ID, into the HBase database. For indexing an entity, AnormStore parses the XML representation of the entity using the Xerces SAX parser [1] and creates a document that temporarily stores all index able fields of an entity. This document is then passed to the SolrMaster for indexing. The SolrMaster to be used by AnormStore API is known from the SolrBroker. Certain fields from the XML entity can be blocked for indexing by the security layer of the Okkam.

3.2.1.2 Deletion of an Entity

One can delete an entity by specifying its Okkam ID to the AnormStore API. AnormStore API provides a single delete method which takes the Okkam ID as an argument and deletes the entity from Solr Index and HBase.

AnormStore API gets the SolrMaster IP address from the SolrBroker, since only the SolrBroker knows the particular SolrMaster where the entity is stored. AnormStore API will later delete the entity from that SolrMaster. SolrSlaves will update their indices by pulling the new snapshot from the SolrMaster, while the HbaseMaster handles the deletion of all the copies of an entity from its slaves.

3.2.1.3 Update of and Entity

One can update also an entity. In this case one has to specify ID of the entity to be updated.

3.2.1.4 Querying the Entities

One has to specify a query using the Lucene query language [2]. In AnormStore, once the query is specified, AnormStore API firstly contacts the SolrBroker to get the SolrSlaves so that the entire repository is covered. AnormStore API then passes the query to each of the SolrSlaves. All
the results (Okkam IDs and scores) from each of the SolrSlaves are accumulated by AnormStore API. These results are ranked based on the scoring function used by Lucene [3]. Currently we are using a standard TF-IDF scoring function extended with the Okkam-specific boosting of attributes and values. The number of results to be returned can be specified in the AnormStore’s configuration file. Once the Okkam IDs are obtained, AnormStore API retrieves the profiles of all entities from HBase by specifying the Okkam ID as the key.

3.2.2. Example

The following simple code example explains the typical usage of the API functions.

```java
public void test(){
  try {
    // Instantiate the configuration. If you wish to change the
    // configuration file then pass a different filename.
    // This is a key step. Since it setups parameters used by the
    // system.
    Configuration conf = Config.getDefault();
    Iterator it = conf.getKeys();
    while (it.hasNext()){
      String key = (String)it.next();
      System.out.println(key + " : " + conf.getString(key));
    }
    // Create a new entity writer.
    IEntityWriter writer = new EntityWriter();
    // Create a new entity searcher.
    IEntitySearcher searcher = new EntitySearcher();
    // Create a new entity reader
    IEntityReader reader = new EntityReader();
    // Filename that contains the entity in XML format.
    String filename = "ok0204ee7e-74af-4e63-8ca8-b4b93133f956.xml";
    // Reading contents of the file which holds the entity.
    String contents = ";
    BufferedReader br = null;
    try{
      br = new BufferedReader(new FileReader(filename));
      String str;
      while ( (str = br.readLine() )!= null)
        contents += str;
      br.close();
    }catch(Exception e){
      System.out.println(e.getMessage());
    }
    // Extract the EntityID from the filename.
    // CAUTION: entityID should be the complete ID:
```

OKKAM - 215032

Version 13.0

Page 20 of 37
String entityID = filename.substring(0, filename.lastIndexOf("."));
String entityID = "http://www.okkam.org/entity/ok0204ee7e-74af-4e63-8ca8-b4b93133f956";

// Create new Entity by giving the entity ID and XML contents.
IStorageEntity entity = new Entity(entityID, contents);

// Write entity to the disk store. This indexes the entity using Lucene and also stores it as a string in Hbase with entityID as a key.
writer.write(entity, null);

// Update an entity. Replaces an old entity with a new one.
//writer.update(entity.getID(), entity);

// Query for "california". This supports keyword queries and Boolean queries like Lucene.
// For more information look at http://lucene.apache.org/java/2_3_2/queryparsersyntax.html
IStorageQuery query = new Query("california");
Vector<IScoredEntity> entityVector = searcher.search(query, null);
int size = entityVector.size();
for (int i = 0; i < size; i++) {
    IScoredEntity se = entityVector.get(i);
    System.out.println("Entity ID : " + se.getID());
    System.out.println("Entity Score : " + se.getScore());
    System.out.println("Entity Contents : " + se.getXMLEntity());
}

System.out.println("\nGet alternative IDs");
System.out.println(reader.getAlternativeIds("http://www.okkam.org/entity/ok0204ee7e-74af-4e63-8ca8-b4b93133f956", null));
System.out.println("\nGet Okkam IDs by alternative ID");
System.out.println(reader.getOidsByAlternativeId("08008335", null));

} catch(OkkamCoreException e) {
    System.out.println(e.getMessage());
}
3.3. Matching API

This section describes the objects and methods defined in the OKKAM Match component that have to be used for employing the OKKAM Match functionality. Section 3.2.1 explains how to request for entities, whereas Section 3.2.2 describes the format of the results of such an entity request.

3.3.1. Requesting Entities

The core functionality provided by the Matching API is answering entity requests. This means it accepts a description of an entity, checks if the described entity is available in the OKKAM entity repository and, if this is the case, returns one or more matching candidates. The entity description is given in the form of an entity request following the rules of the OKKAM Request language. The matching candidates are those entities in the repository that according to the employed matching algorithm are possible matches for the entity described by the entity request.

3.3.1.1 Methods for Requests

The OKKAM Match component offers two main methods for requesting entities (Table 1 provides the signatures of these methods):

- **Single Resolution Request.** This method is used for retrieving the entities (matching candidates) that match the entity request. The entity request is given as a string. The result of a single resolution request is a matching candidates bundle; a ranked list with entities that have been matched to the entity request.

- **Bulk Resolution Request.** This is a method for requesting the resolution of a collection of (interlinked) entities. Similar to the single resolution request, each entity is specified by a request, and the result is a collection of matching candidate bundles.

```java
IMatchingCandidateBundle getEntity (String stringQuery, ISessionMetadata sessionMetadata);
Collection<IMatchingCandidateBundle> getEntities(List<String> stringQueryCollection, ISessionMetadata sessionMetadata);
```

Table 1. The main methods available in OKKAM Match for making entity requests.

In the next paragraphs we describe the format of the requests itself (Section 3.2.2.2) and how to select a matching module for a request (Section 3.2.2.3).

3.3.1.2 Formulating Requests

Formulating requests is done using the OKKAM Request language which is described by the grammar in Table 2. A full request consists either of a query, or a query together with metadata and context. Metadata is used for selecting the matching module (see Section 3.2.1.3).

A well-formed query consists of (eventually nested) conjunctions and disjunctions of words or URIs (interpreted as values) to which following information can be added:
- relevance of the value within the whole query, and
- an attribute for this value.

```plaintext
/*** Atomic tokens ***/
OR ::= "OR"
AND ::= "AND"
OPERATOR ::= "="
URI_LEFT_DELIMITER ::= "<"
URI_RIGHT_DELIMITER ::= ">
RELEVANCE_OPERATOR ::= "*"
LEFT_PARENTHESIS ::= "(
RIGHT_PARENTHESIS ::= ")"
LEFT_CURLY ::= "{"
RIGHT_CURLY ::= "}"

/*** Structured tokens ***/
ATTRIBUTE_NAME_OR_VALUE ::= Nmtoken | RELEVANCE | QuotedString
DELIMITED_URI ::= URI_LEFT_DELIMITER URI URI_RIGHT_DELIMITER
RELEVANCE ::= float

/*** Query ***/
QUERY_SECTION ::= "QUERY" LEFT_CURLY QUERY RIGHT_CURLY
QUERY ::= TERM (OR QUERY)*
TERM ::= NOT? FACTOR (AND? TERM)*
FACTOR ::= (ATTRIBUTE_NAME_OR_VALUE OPERATOR)? (
ATTRIBUTE_NAME_OR_VALUE | DELIMITED_URI)
(RELEVANCE_OPERATOR RELEVANCE)? | LEFT_PARENTHESIS QUERY RIGHT_PARENTHESIS
```

Table 2. Request language grammar.

The following are three examples of requests which describe Einstein, the famous Nobel prize physicist:

- **R.1:** name = “Albert Einstein”
- **R.2:** Albert Einstein
- **R.3:** name = “Albert Einstein” Nobel Prize

### 3.3.1.3 Selecting Matching Module

A request can be also accompanied with metadata and context information. Metadata are used for selecting the matching module to be used during the processing of the request. The underlying matching framework provides the option to select matching modules at processing time. Context information is currently not used but provided for future extension. Table 3 provides the respective part of the request language grammar.
Table 3. Request language grammar (matching module selection).

The following are two examples of requests which define the matching module that should be used for processing:

- **R.1**: `QUERY { name = "Albert Einstein" } METADATA {matchingModule=glinkage}
- **R.2**: `QUERY { Albert Einstein} METADATA {matchingModule=glinkage}

### 3.3.2. Result format

The result of each request is a list of candidate entities found currently in OKKAM, which were matched with the given request, i.e., which satisfy the conditions described in the request. The number of entities in this return list is ideally only one, since each request is made for retrieving a specific entity. As such, OKKAM Match returns an IMatchingCandidateBundle object that contains the matching entity candidates. The following table shows the methods available under this object.

<table>
<thead>
<tr>
<th>IMatchingCandidateBundle</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterator&lt;IMatchingCandidate&gt;</td>
<td>An iterator for navigating over the results of the specific request.</td>
</tr>
<tr>
<td>iterator();</td>
<td></td>
</tr>
<tr>
<td>int size();</td>
<td>Gets the size of the bundle.</td>
</tr>
</tbody>
</table>

#### 3.3.2.1 Matching Candidates

A Match candidate encodes an entity matched with a specific request. It is given a IMatchingCandidate object and it allows to access three related information, the OKKAM identifier of the specific entity, the entity profile, and matching probability computed during matching. The following table shows the methods available under this object.

<table>
<thead>
<tr>
<th>IMatchingCandidate</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getOkkamId();</td>
<td>Gets the matching candidate profile.</td>
</tr>
<tr>
<td>Entity getEntityProfile()</td>
<td>Gets the OKKAM identifier of this matching candidate.</td>
</tr>
<tr>
<td>double getProbability();</td>
<td>Gets the matching probability.</td>
</tr>
</tbody>
</table>
3.3.2.2 Match probability

Matching probability depends on two main factors: similarity and distinctiveness. If the values of a request attribute and an entity attribute are similar (w.r.t. an underlying error-model), the match probability increases. But in addition, a match on an attribute with fairly unique values, such as name, contributes much more evidence for a match of the entity than a match on an attribute with very few values, such as gender = male or female.

More formally, this is captured by the Fellegi-Sunter criterion for duplicate detection. Adapted to the OKKAM scenario the criterion can be stated as follows:

\[
(r, e) \in \begin{cases} 
M, & \text{if } P(M|r, e) > P(U|r, e) \\
U, & \text{otherwise}
\end{cases}
\]

This criterion says that entity e should be accepted as a match for request r, if the probability for a match M given (r; e) is larger than the probability for no match U given (r; e).

The method getProbability under each Matching Candidate returns the matching probability for this entity given the respective request.

3.4. Entity Lifecycle API

This section aims at providing an overview of the objects and the methods defined in the Lifecycle Manager Component in order to manage entity creation and update tasks, entity description default attributes and other auxiliary functions. In section 3.4.1 we give an overview of entity creation methods. Section 3.4.2 gives a brief description of the entity update operation. Finally section 3.4.3 provides a description of some auxiliary functions currently implemented within lifecycle manager component.

3.4.1. Creating a New Entity

One of the core functionalities of the currently implemented lifecycle manager component is the creation of new entity in the ENS. A proper management of this operation is essential to guarantee minimum number of entity duplicates in the ENS, an adequate quality of entity description. With this purpose, in the context of entity creation, a check for duplicates is performed every time before actually writing a new entity in the ENS, and entity description default attributes are made available for the external application.

3.4.1.1 Methods for Default Attributes management

The ENS currently supports a set of Entity Semantic Types aiming at defining a high level categorization of the entities within the ENS. One of the goals of this categorization is to provide for each entity type a set of default attributes which can be used to create a core template for the

---

7 Fellegi and Sunter formalized this criterion, aka “Bayes Decision Rule for Minimum Error”
entity description. The aim of the default attributes associated with each entity type is to support the human user in defining unambiguous descriptions of the entity at creation time.

The method supporting the retrieval of the supported entity types is:

\[\text{List<String> getEntityTypes()}\]

The purpose of this method is to return a list of all the entity types currently supported by the ENS.

The method supporting the retrieval of the default attributes associated with each supported entity type is:

\[\text{List<String> getDefaultAttributes(String entityType)}\]

The purpose of this method is to support external applications in dealing with the default attributes, by returning a list of the default attributes of the specified entity type.

### 3.4.1.2 Methods for Entity Creation

The lifecycle manager component supports 2 ways to create new entity in the ENS: single entity creation and batch entity creation.

**Single Entity Creation**

*Using single entity creation a user can create a (single) new entity using the createEntity() method.*

\[\text{NewEntityResult createEntity(Entity entity, boolean force, ISessionMetadata m)}\]

The purpose of this method is to the procedure for the creation of a new entity. The parameters are the new 'entity' object to be created, a Boolean parameter 'force' indicating whether entity creation should be forced against duplicate detection, and session metadata 'm'.

This process goes through a duplication check. Duplication check is basically performed by composing a query using the description of the under creation entity, and submitting such query to the matching module through the \text{getMatches()} method.

\[\text{List<Entity> getMatches(Entity entity, ISessionMetadata sessionMetadata)}\]

The purpose of this method is to check whether the specified entity contains duplicates in the ENS. It returns a list of entities similar to the given entity.

The list of entities matching the entity description is then filtered according to a threshold value (currently 0.67). All the entities with similarity below the pessimistic threshold are discarded as duplicate candidates.

If the entity creation is completed successfully, the list of candidate duplicates is empty. If the duplication check detects candidate duplicate entities, the new entity identifier parameter is set to 'null' and the list of candidate duplicates returned is filled. If the 'force' parameter is set to true, the entity creation process skips the check for duplicates and directly creates and write the new entity in the ENS.
Batch Entity Creation

Batch entity creation does not perform any duplication check because it is basically used only in the context of initial entity import, thus the no duplicates' existence is assumed.

```java
List<NewEntityResult> createEntities(List<Entity> entities, boolean force, ISessionMetadata sessionMetadata)
```

The purpose of this method is to support the creation of a group of entities in sequence.

Entity batch import passes through three basic steps: 1) state the beginning of the import operations invoking the method `batchImportBegin()`, 2) iterated over the list of entities to create invoking for each entity the method `batchImport()`, 3) state that the import process is complete invoking the method `batchImportCommit()`.

In case the imported entities is carrying a valid entity identifier, it is possible to execute a batch import step without overwriting the entity identifier by invoking the method `batchImportKeepOkkamIds()` on place of the `batchImport()` method. This method basically executes the batch import of the entity preserving the entity identifier passed with the 'entity' parameter.

### 3.4.2. Update Entity

Another important functionality implemented in the lifecycle manager component is the update of an entity description. The current business logic behind the entity update method is to retrieve the former entity description, merge it with the updated one and then check whether the new description makes the entity more similar to other existing entities by performing a duplication check.

```java
UpdateEntityResult updateEntity(Entity entity, boolean force, ISessionMetadata sessionMetadata)
```

The purpose of this method is to perform the update of an entity description, and the Boolean parameter 'force' indicates whether entity creation should be forced against duplicate detection.

If the entity update is completed successfully, the list of candidate duplicates is set to 'null'. If the updated entity description makes the entity similar to other existing entities, the list of candidate duplicates is returned.

Entity update basically retrieves the original entity in the ENS, performs a merge between the new entity description and the original one, and checks whether the result of entities' description merging is similar to other existing entities.

As of April 2009, the functions for entity merging are not yet implemented. Their precise functionality will be finalized shortly.

### 3.4.3. Auxiliary methods

EntityCreator is a web application supporting users in creating new entities. EntityCreator can be used by a client application as “external application” to fulfill entity creation tasks, thus we need a mean to enable communication between EntityCreator and the client application. The current solution to enable communication via the ENS The process is the following:
1. The client application gets a session metadata by opening a session in the ENS.
2. The client application invokes EntityCreator passing the session metadata as parameters, now EntityCreator and the client application share session information.
3. The user completes the entity creation process through EntityCreator, the last stores the newly created entity in the lifecycle manager component by invoking the method `selectEntity()` using the passed session metadata information.
4. The user resumes the client application which gets the newly created entity by invoking the method `getSelectedEntity()` with the same session metadata information passed to EntityCreator.

The `selectEntity()` method couples the newly created entity with session metadata information by storing them in a hash-table. The session metadata information is used as key in the hash-table.

The `selectSelectedEntity()` method retrieves the newly created entity by returning the entity object associated with the key composed by the session metadata information.

If the entity creation process wasn't completed correctly or the operation is still under execution, the created entity field of the returned object is set as 'null'.

### 3.5. Security API

This section describes the functional aspects of OKKAM security APIs, as of April 2009. The section does not reflect the planned changes related to adapting stateless connections instead of using stateful sessions and the extensions to support REST services.

#### 3.5.1. Overview of authentication and access control

Below we summarize the core authentication and access control functionality and how they will be handled via the security APIs. Authentication and access control are managed via a user-centric proxy component. Figure 1 shows the proxy model and its high-level message flow.

![Security proxy model and its message flow](image)

The adoption of the proxy model is: (i) to decouple potential OKKAM applications from security logic necessary for accessing ENS services, (ii) to provide transparent management of security settings regardless of the type OKKAM applications (e.g., thin applications like MS Word plug-ins, or thick clients like standalone applications), (iii) to increase security by automated credential-based authentication and access control mechanism.
3.5.2. Functionalities of the proxies

There are two main functionalities the proxy provides:

- Direct or indirect via a proxy communications with ENS services allowing for transparent switch from unsecure to secure communications (and vice versa). Essentially, all direct communications between OKKAM applications and ENS services are non-secure public communications and, therefore, successful when allowed by the ENS access manager layer. For example, results of a query process via direct requests contain only identifiers with unprotected attributes.

- On-the-fly authentication and access control as fully transparent background processing. This allows OKKAM applications to be freed from the need to include authentication and access control mechanisms, thus facilitating homogeneous control of security settings by providing a single point of management. The approach increases efficiency of access control by managing (reuse) of a security token among different applications run by a same user. In this way, developers of OKKAM applications are abstracted from taking care of security settings that ENS services require.

The proxy component will implement all ENS public APIs as locally accessible services, even those APIs that do not require any protection will be provided for compatibility issue. In this way, an OKKAM application can achieve secure access to ENS services via a change to local host interactions but without changing its internal logic.

3.5.3. Interaction with the proxies

Figure 2 presents in details how the proxy component interacts with ENS APIs, high-level protocol steps and respective messages.

Figure 2: Security proxy protocol interactions with ENS APIs
An OKKAM application wants to have a secure access to an ENS_Service and invokes the corresponding ENS_Service API locally at the proxy. The proxy checks if the invoked service needs protection and what type of access control is required, in order to follow the respective protocol steps.

There are two main access control processes for ENS services: access control before ENS service execution, and access control process after ENS service execution. For example, findEntities() API has access control after its execution in order to filter out those protected attributes (of selected identifiers) that the end user is not qualified to see/know about them.

In case the invoked service requires no protection/security the proxy bypass all protocol steps and just invokes the respective API at the ENS side and returns the response back to the application.

Once the proxy identifies the security settings for the requested ENS service, it follows the four general phases as illustrated in the figure above.

1. **Mutual authentication** of the proxy and the ENS cluster of the domain the proxy is configured to connect to. Note that this phase runs without sticking the interactions to a specific node of the cluster. There are two security APIs envisaged as part of the OKKAM public APIs for this phase: getServerCertificate() and establishSessionKey().
   - getServerCertificate() returns the public key certificate of a node in the cluster. All ENS nodes in a cluster will have a same public key certificate (they are replicas).
   - establishSessionKey() serves to establish a session key out of mutual authentication. We note that this function for the time being is automatically implemented by the METRO\(^8\) Web service library stack and may not appear as an explicit API of ENS services. When a move to REST Web services then this function may appear explicitly for use by the proxy part. We visualize this function to emphasize on the session ID/Session key generated internally by an ENS node but used throughout the messages of the proxy protocol.

2. **WS service forwarding** phase where the proxy invokes the ENS_Service at the ENS side. In curl brackets we denote additional messages (staying in the optional header element of the SOAP envelope) necessary for security communications. In this request the proxy includes a proxy flag indicating that the request comes from a proxy component, and a security token indicating already presented access rights to the ENS system. The security token has the same purpose as the cookies for Web browsing, but has a specific format and structure (possibly a SAML token).

   If more security is necessary for the given WS request the response to the proxy includes a flag in the header denoting that (shown in the figure as need_more_security).

3. **Interactive access control phase** for establishing the missing access rights for accessing/visualizing the requested service/query results. There is a dedicated API at ENS (shown in the figure as runNegotiation()) for upgrading the missing access rights. This phase includes a negotiation process for satisfying user and ENS credential requirements. The result of this phase is an updated security token (or a new one if it does not exist) attesting the user presented credentials. The result of the final message of this phase may also include all query results visible to the user based on the newly agreed credentials. The latter case is

\(^{8}\) https://metro.dev.java.net/
for optimization purposes to avoid, for example, executing already executed matching process, in case access control takes place after execution.

4. *WS message forwarding* phase is repeated for those ENS_Service requests that require access control before a service execution, so at this step the proxy invokes the originally requested ENS_Service but with the new security token from the last negotiation phase. No further interactions within the established session are repeated in case a flag for more security is returned.

The above presented proxy-based authentication and access control process serves to define the main protocol steps and their interactions with dedicated for each step ENS APIs. Some of the security APIs may be subject to change towards the release of ENS v2.0, especially the input message types in order to strictly conform to the recent decisions/changes to the ENS v2.0 architecture.
4. Conclusions and future plans

This deliverable contains a guide to the public and private OKKAM APIs. The guide reflects the functionality of the APIs as of April 2009 (the contractual date of delivery of this document). As the software might further evolve (and indeed it does), we will consider revising the individual chapters at a later stage of the project. We plan to revise the API guide after major releases only; however we intend to keep the javadoc documentation always up to date through regular and automatic updates.

As explained in section 1.1, the automated generation of OKKAM API and Technical Documentation is only beginning at ENS V2.0. So the API descriptions provided in this report have been mainly generated manually. However as the documentation commenting standards specified in section 1.1 are adopted and used in the various components of the OKKAM system, the core content of this API Reference will be automatically updated at each minor release of the OKKAM System and available at http://gforge.okkamdev.org/TechnicalDocs/index.html.

The documentation commenting standards specified in section 1.1 are adopted by the majority of components already, but it is expected that by the release of ENS V3.0 it will be the norm. The up-to-date technical documentation, which will continuously available at http://gforge.okkamdev.org/TechnicalDocs/index.html to the OKKAM Development Community, shall also reinforce the adoption of documentation standards. This, and its use within the overall OKKAM Development Community into the future beyond the current project, will be documented in the deliverable D5.7 (OKKAM Developers Guide) at the end of this project.
5. References

Annex A: javadoc

Javadoc is a documentation generator from Sun Microsystems for generating API documentation in HTML format from Java source code\(^9\). The "doc comments" format used by Javadoc is the de facto industry standard for documenting Java classes. Javadoc also provides an API for creating doclets and taglets, which allows you to analyze the structure of a Java application.\(^{10}\)

A.1 How to Write Doc Comments for the Javadoc Tool\(^{11}\)

The Sun Microsystems website at [http://java.sun.com/j2se/javadoc/writingdoccomments/index.html](http://java.sun.com/j2se/javadoc/writingdoccomments/index.html) describes the style guide, tag and image conventions that they use in documentation comments for Java programs that are immediately relevant to OKKAM Developers, who mainly program in Java.

There are commonly two different ways to write doc comments:

- API specifications, which is the focus here, to give prime focus to writing API specifications in doc comments.
- Programming guide documentation – which is covered in D5.1.

Ideally, the OKKAM API Specification comprises all assertions required to do a clean-room implementation of the OKKAM Platform for "write once, run anywhere" -- such that any OKKAM applet or application will run the same on any implementation. This may include assertions in the doc comments plus those in any architectural and functional specifications or in any other document. However in reality there are some practical limitations to how fully Developers can specify the API. The following guiding principles, which are the best practice used by Java Software at Sun Microsystems, are those recommended for the OKKAM system:

1. Each OKKAM Platform API Specification is defined by the documentation comments in the source code and any documents marked as specifications reachable from those comments.
   - So the specification does not need to be entirely contained in doc comments. In particular, specifications that are lengthy are sometimes best formatted in a separate file and linked to from a doc comment.
2. Each OKKAM Platform API Specification is a contract between callers and implementations.
   - The Specification describes all aspects of the behavior of each method on which a caller can rely. It does not describe implementation details, such as whether the method is native or synchronized. The specification should describe (textually) the thread-safety guarantees provided by a given object. In the absence of explicit indication to the contrary, all objects are assumed to be "thread-safe" (i.e., it is permissible for multiple threads to access them concurrently). It is recognized that current specifications don't always live up to this ideal.

\(^{11}\) At [http://java.sun.com/j2se/javadoc/writingdoccomments/index.html](http://java.sun.com/j2se/javadoc/writingdoccomments/index.html)
3. Unless otherwise noted, each OKKAM API Specification assertions need to be implementation-independent. Exceptions must be set apart and prominently marked as such.

4. Each OKKAM API Specification should contain assertions sufficient to enable Software Quality Assurance to write complete JUnit\(^\text{12}\) system tests.
   - This means that the doc comments must satisfy the needs of the conformance testing by SQA. The comments should not document bugs or how an implementation that is currently out of spec happens to work.

As the OKKAM Development Community is expected to evolve and extend\(^\text{13}\) these guidelines are intended to describe the finished documentation comments. They are intended as suggestions rather than requirements to be slavishly followed if they seem overly burdensome, or if creative alternatives can be found. When a complex system such as OKKAM (which contains many components) is being developed, often a group of engineers contributing to a particular component, may develop guidelines that are different from other development groups and organisational settings. This may be due to the differing requirements of those components, or because of resource constraints.

**A.1.1 Who Owns and Edits the Doc Comments**

The doc comments for each OKKAM platform API specification is initially owned by the programmers. However, we expect that as the OKKAM Development Community develops and spreads, they will be edited by both programmers and writers. However it will remain a basic premise that writers and programmers honour each other's capabilities and both contribute to the best doc comments possible. Often it will be a matter of negotiation to determine who writes which parts of the documentation, based on knowledge, time, resources, interest, API complexity, and on the state of the implementation itself. But the final comments must be approved by the responsible engineer.

The person designing the API should always write the API specification in skeleton source files, with only declarations and doc comments, filling in the implementation only to satisfy the written API contract. The purpose of an API writer will be to relieve the designer from some of this work. In this case, the API designer should write the initial doc comments using sparse language, and then the writer review the comments, refine the content, and add tags.

**A.2 Requirements for Writing OKKAM API Specifications\(^\text{14}\)**

The Sun Microsystems website at [http://java.sun.com/j2se/javadoc/writingapispecs/index.html](http://java.sun.com/j2se/javadoc/writingapispecs/index.html) describes the requirements for writing API specifications for the Java platform, which again are directly relevant to OKKAM Developers working in Java. In keeping with this best practice, the specification for each OKKAM platform API library is made up of its Javadoc comments and additional support documentation linked out in the doc comments (as described above).

---

\(^\text{12}\) See [www.junit.org](http://www.junit.org)

\(^\text{13}\) See the D5.1 Deployment Guidelines.

The website describes a framework of five sections, of which sections 1, 2 and 5 correspond to the following sections of an OKKAM API specification; each section (except the first) includes examples which can be directly related to OKKAM.

   a) Top-Level Specification
   b) Component Specification
   c) Method Specification

While the website gives details of all 5 sections, the following is a summary these 3 sections to see how to map to each other.

The basis of the API specifications is Assertions: An **assertion** is a testable statement that specifies some necessary aspect of the API. You can think of an assertion as a specification that a clean-room implementer would have to follow in order to implement the OKKAM platform. An assertion is a specification that application developers can rely upon. For example, the statement "Returns an int" is an assertion. A code example is not an assertion. Assertions are critical to conformance testing and implementers of the OKKAM Platform.

### A.2.1 Top-Level Specification

The top-level specification is composed of those specifications that apply to the entire set of OKKAM components. It can include assumptions that underlie the other specifications, such as all objects are presumed to be thread-safe unless otherwise specified.

### A.2.2 Component Specification

The Components specification includes any specifications that apply to an OKKAM Component as a whole or to groups of classes in the Component. It must include:

   a) Executive summary - A precise and concise description of the Component. Useful to describe groupings of classes and introduce major terms.
   b) OS/Hardware Dependencies - Specify any reliance on the underlying operating system or hardware.
   c) References to any external specifications. These are Component-wide specifications beyond those generated by Javadoc or third-parties. These references can be links to specifications published on the Internet, or titles of specifications available only in print form. The references must be only as narrow or broad in scope as the specification requires. That is, if only a section of a referenced document is considered part of the API spec, then you should link or refer to only that section (and can separately refer to the non-spec of the document as a "related" document). The idea is to clearly delineate what is part of the API spec and what is not.

### A.2.3 Method Specification

This section applies to OKKAM system methods and constructors. Each method and constructor specification must include:

1. Expected Behavior - Specify the expected or desired behavior of this operation. Describe what aspect of the object being modeled this operation fulfills.
2. State Transitions - Specify what state transitions this operation may trigger.
3. Range of Valid Argument Values - Specify all valid and invalid values for each argument, including expected behavior for invalid input value or range of values.

4. Null Argument Values - For each reference type argument, specify the behavior when null is passed in. NOTE: If possible, document the general null argument behavior at the Component level, such as causing a NullPointerException to be thrown. Deviations from this behavior can then be documented at the method level.

5. Range of Return Values - Specify the range of possible return values, including where the return value may be null.

6. Algorithms Defined - When required by the specification, specify the algorithms used by this operation.

7. OS/Hardware Dependencies - Specify any reliance on the underlying operating system or hardware.

8. Allowed Implementation Variances - Specify what behavior may vary by implementation. This description should not include information about current OKKAM System bugs.

9. Cause of Exceptions - Specify the exceptions thrown by the method, include the argument values, state, or context that will cause the specified exception to be thrown. The exceptions thrown from a method need not be mutually exclusive.

10. Security Constraints - If this operation may be security constrained, must specify the security check used to constrain this operation. Include a general description of the context or situations where this method may be security constrained.