### DOCUMENT HISTORY

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
<th>Action(*)</th>
<th>Confidentiality</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>23/12/2011</td>
<td>IDS</td>
<td>Creation</td>
<td>C</td>
<td>High</td>
<td>7</td>
</tr>
<tr>
<td>1.1</td>
<td>18/07/2012</td>
<td>IDS</td>
<td>Changed template: EC 2012</td>
<td>U</td>
<td>High</td>
<td>NA</td>
</tr>
<tr>
<td>1.2</td>
<td>07/12/2012</td>
<td>IDS</td>
<td>Adapt list of default published attributes</td>
<td>U</td>
<td>High</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Adapt introduction</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(*) Action: C= Creation, I=Insert, U=Update, R=Replace, D=Delete
TABLE OF CONTENTS

1. Introduction .............................................................................................................................................. 5
   1.1 Purpose of this document ........................................................................................................... 5
   1.2 Usage .................................................................................................................................................. 5
       1.2.1 Overview ................................................................................................................................... 5
       1.2.2 Enabling metadata publishing ................................................................................................. 5
       1.2.3 Sample use case ....................................................................................................................... 5
   1.3 Set up and initialization ................................................................................................................... 5
2. Useful Queries ........................................................................................................................................... 7
   2.1.1 Why using hash codes? ............................................................................................................. 7
   2.1.2 Terminology: translations ........................................................................................................ 7
   2.1.3 Languages joins: efficiently retrieving every translation of a document ......................... 7
3. Annex: MurmurHash3 algorithm .......................................................................................................... 9
LIST OF TABLES

- TABLE 1: COMPUTED ATTRIBUTES ........................................................................................................ 6
1. **INTRODUCTION**

1.1 **Purpose of this document**

This document contains a user manual for the CWCM Metadata Publishing features.

1.2 **Usage**

Since version 3.2.0.0, the CWCM publication mechanism allows publishing documents metadata to tables into a relational database management system known as the dissemination RDBMS.

1.2.1 **Overview**

Custom external dynamic applications can query the dissemination tables to build on the fly pages based on result sets returned by SQL queries.

Metadata publishing is enabled per sub site, and is disabled by default.

The dissemination tables allow identifying a document by its unique URL (stored as a_webc_url) and metadata for this document are “attached” to this URL.

1.2.2 **Enabling metadata publishing**

End users and application developers must request the Comm Europa Management (a.k.a. CEM) helpdesk for an access to the dissemination tables and/or for enabling metadata publishing on a specific subsite. See “1.3 Set up and initialization”.

1.2.3 **Sample use case**

Metadata publishing features may allow retrieving a set of URL’s based on some criteria (such as language, category, subsite and so on depending on list of published metadata).

The custom external dynamic applications may use the retrieved URL’s to generated link boxes on the fly, such as related links boxes for instance, on the dynamic page.

1.3 **Set up and initialization**

The metadata publishing itself is set up and maintained by DIGIT Data Centre.

The following items are to be negotiated with the Data Centre, via the COMM EUROPA MANAGEMENT:

- RDBMS credentials;
- List of sub sites for which metadata publishing must be enabled;
- Table or view names, structure and indexes;
- List of metadata to be published.

Table or view provided by the Data Centre can be different from what the system actually publishes. The underlying system creates two tables, one for single attributes and one for repeating attributes, and the join is done using a_webc_url.
Single and repeating attributes may be split to several views or not. The CWCM documentation (SA.custom.doc) provides list of documents metadata that can be published.

Everything described above must be checked and discussed with Data Centre, since some specific views might be generated for your sub site. Also the implementation of an additional layer, as a web service, should be available at a certain stage: the date of availability will be provided by the CEM.

At least the following metadata are always provided:

- a_webc_url
- a_webc_hash
- translation_path
- translation_hash

These attributes are computed by the metadata publishing mechanism, i.e. they do not natively exist in the CWCM repository.

These additional data are keys to retrieve a document or a set of documents.

The “a_webc_url” attribute is the path to the document’s location within the published site.

The “a_webc_hash” attribute is the hash code, based on MurmurHash3 algorithm, for a_webc_url.

The “translation_path” attribute is equal to “a_webc_url” with “zz” instead of the language extension.

The “translation_hash” attribute is the hash code, based on MurmurHash3 algorithm, for translation_path.

The computed attributes are stored as follows:

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Data type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_webc_url</td>
<td>varchar2</td>
<td>544</td>
</tr>
<tr>
<td>a_webc_hash</td>
<td>Integer</td>
<td>N/A</td>
</tr>
<tr>
<td>translation_path</td>
<td>varchar2</td>
<td>544</td>
</tr>
<tr>
<td>translation_hash</td>
<td>Integer</td>
<td>N/A</td>
</tr>
</tbody>
</table>

- **Table 1: Computed Attributes**

We explain how to use these attributes in chapter “2 Useful Queries”.

The MurmurHash3 algorithm is described in chapter "3 Annex: MurmurHash3 algorithm”.

Some other attributes are also always published, for a technical purpose, and should not be used by external applications. For information, these attributes are: r_object_id, i_chronicle_id, object_name, r_modify_date.

All other published attributes depend on the configuration set up by the Data Centre, and the information is provided by CEM.
2. Useful Queries

This paragraph describes how to use the following computed metadata:

- `a_webc_url`
- `a_webc_hash`
- `translation_path`
- `translation_hash`

We do not use any other attributes, since the other ones are either published for pure technical reasons, or are fully configurable by the data centre. We only discuss in the frame of a direct database access, i.e. not using any additional layer such as a web service.

An imaginary document `welcome_fr.htm`, from the domain ec.europa.eu and subsite msa, will be published with the following associated metadata:

```
A_WEBC_URL: msa/welcome_fr.htm
A_WEBC_HASH: -1778034461
TRANSLATION_PATH: msa/welcome_zz.htm
TRANSLATION_HASH: -839195464
```

Other metadata might have been published as well depending on the configuration associated to the subsite.

2.1.1 Why using hash codes?

The `a_webc_url` and `translation_path` columns are `varchar2` type. When these fields are part of a `WHERE` clause this means comparison of long string is going to occur. This is not efficient in term of performances.

For this reason, we prefer to provide the associated hash code for each string, using the MurmurHash3 algorithm. This allows retrieving every row with the same hash code using Integer comparison, which is much more efficient than string comparison. Some different strings may have the same hash code though, therefore it is still necessary to perform a string comparison, but on a smaller set. So we advice using a `WHERE` clause based on hash code rather than on string value. The string value comparison can be added as a second clause.

These hash codes can be used when retrieving every translation of a document for instance.

2.1.2 Terminology: translations

To understand better how to use the publishing metadata tools, we need ensuring we use the correct definitions for translations.

A translation of a document is another document sharing the same path, name and file extension in the source repository (CWCM repository) but having a different language extension. For instance, in ec.europa.eu/msa, `index_fr.xml` would be a translation for `index_en.xml`.

2.1.3 Languages joins: efficiently retrieving every translation of a document

In order to get a join between each translation of a rendition, we create a “translation_path” attribute that is equal to “a_webc_url” with “zz” instead of the language extension.
For the translation path, including the language extension zz to the names is necessary because index.htm and index_en.htm can both exist in the same sub site and are unrelated.

The hash code allows accelerating performances to retrieve every translation of a file. The hash code cannot be used as a key, because its uniqueness is not guaranteed; the client applications always need to check both the hash and the string value to be sure of the translation relationship between the objects.

A query allowing retrieving fast the available languages would be:

```sql
select SI.A_WEB_URL from SCHEMA.TABLE_NAME SI
where SI.TRANSLATION_HASH = 190620529 and SI.TRANSLATION_PATH = 'rpp-s/samples/chunk1_zz.htm'
```

Remark: if at the certain moment, the performances of the query are bad due to the use of the translation path attribute, it is best to perform the request only using the translation hash then to clean up the result set in the code of the application to only keep the actually requested translation path.
3. Annex: MurmурHash3 Algorithm

To compute hash code for a string (typically a_webc_url and translation_path), we use the MurmурHash3 algorithm. We use the MurmурHash3 implementation in Java, based on Patrick McFarland’s version: http://code.google.com/p/cumulusrdf/source/browse/trunk/src/edu/kit/aifb/cumulus/store/MurmurHash3.java?spec=svn122&r=122.