ECP 2008 DILI 518002 EUscreen

Exploring Europe's Television Heritage in Changing Contexts

D4.8 – Report on semantic interoperability with Europeana

Deliverable number	*D4.8 – Report on semantic interoperability with Europeana*
Dissemination level	*Public*
Delivery date	*12 July 2012*
Status	*Final*
Author(s)	*Vassilis Tzouvaras, Kostas Pardalis, Marco Rendina, Johan Oomen*

e*Content*plus

This project is funded under the *e*Content*plus* programme[1]
a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.

1 OJ L 79, 24.3.2005, p. 1.

Document Information

Deliverable number:	*D4.8*
Deliverable title:	*Report on semantic interoperability with Europeana*
Actual date of deliverable:	M30
Workpackage:	4
Workpackage title:	Semantic Access & Integration
Workpackage leader:	NTUA
Keywords:	Interoperability, Europeana, OAI-PMH

Table of Contents

1 Introduction

Present document constitutes the report for the Interoperability with Europeana. The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) is the chosen technology for exporting metadata items to Europeana.

OAI-PMH is a low-barrier and widely used mechanism for repository interoperability. Data providers are repositories that expose structured metadata via OAI-PMH. Service providers then make OAI-PMH service requests to harvest that metadata. OAI-PMH is a set of six verbs or services that are invoked within HTTP. In the context of the EUscreen project, OAI-PMH provides a mechanism for interoperability between the Ingestion Tool and various other modules or platforms. E.g. using the OAI-PMH terminology, the Ingestion Tool constitutes the data provider while Europeana is able to request and retrieve metadata records via the OAI-PMH verbs or services.

Section 2 illustrates the technical procedure to set up the exporting module. Section 3 presents the mapping between EUscreen elements and the Europeana Semantic Elements. Finally, in Appendix A, the xslt code is included.

2 Interoperability with Europeana

The Ingestion Tool (constructed within the project, see D4.4 Report on EUscreen web services) is capable for managing heterogeneous collections of metadata records while exposing services for mapping and transforming from one metadata schema to another.

In order to extend the functionalities of the Ingestion Tool with the OAI-PMH protocol and thus to expose Metadata through an interoperable mechanism, NTUA has implemented the defined OAI-PMH verbs on top of the underlying and domain-specific data layer. An issue that arises for the case of the Ingestion Tool is that while being able to manage collections of metadata records, the OAI-PMH verbs operate on an item level, something that makes the implementation of the appropriate verbs, directly on top of the Ingestion Tool data layer a difficult task. For this reason and also because it is desired to also include a set of other functionalities that are not directly associated with both the Ingestion Tool and the OAI-PMH protocol, it was decided to follow a technical approach in which an exporting mechanism exist between the Ingestion Tool platform and another data repository more suitable for the needs of an OAI-PMH data repository.

The mechanism for delivering the EUscreen records to Europeana, consists of a data processing layer which is responsible for iterating the records that are stored inside the EUscreen portal index, transform and manipulate them, together with a data layer which is exposed to Europeana by using an implementation of the OAI-PMH protocol. The overall architecture is depicted in Figure 1.

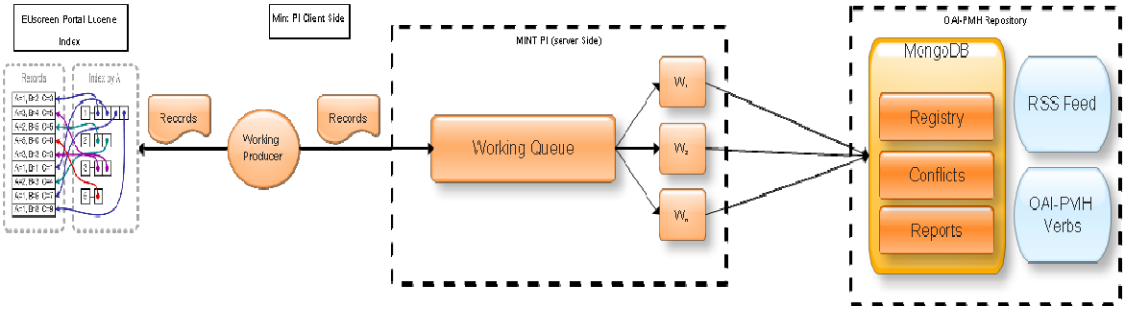


Figure 1: Data Delivery Architecture

MINT PI is part of the Mint interoperability services suite and serves mainly as a scalable mechanism for structured data processing. It is built using RabbitMQ in its core and utilizes a number of message queue patterns as they were described earlier.

The approach of using message queues instead of a mechanism like Hadoop, was decided based on the requirement to scale on both large and small datasets without reducing the efficiency of the overall system. The overall architecture of MINT PI is depicted in Figure 2.

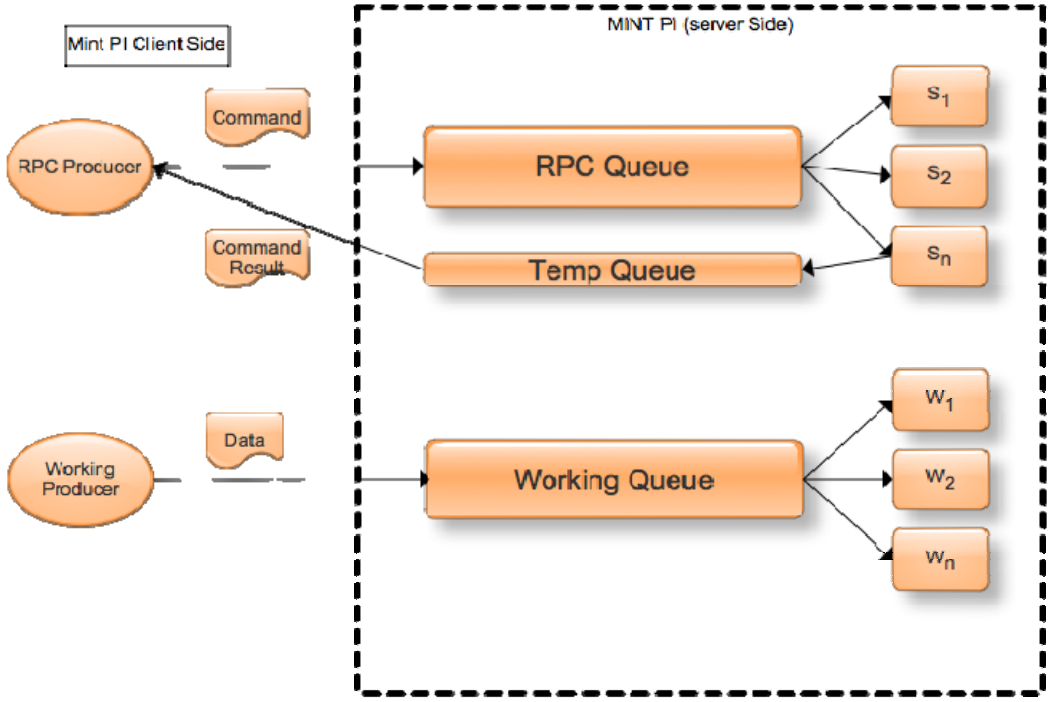


Figure 2: The Mint Processing Interface (PI Architecture)

MINT PI (Processing Interface) is using two distinct queue patterns, an RPC Queue pattern which is used for cases where the client desires to block while the processing is executed and also awaits for a response in a pre-defined format and a Working Queue pattern which is used for non blocking processing where the client submits the data for processing and does not wait for a response. The first case is mainly used for the implementation of specific commands, e.g. for implementing the cleaning or deletion of a repository, while the second case is used for bulk processing of raw data, e.g. data transformation and enrichment of records.

Scalability is achieved by the parallel processing of many workers for the case of the working Queue and the existence of number of RPC Consumers that are running concurrently for the case of the RPC Queue. In both cases the workers and the RPC consumers might be running on different nodes of the cluster that is materialized while more workers and consumers can be added to the system at any time and thus increasing the processing power of the system. At the same time, the RabbitMQ broker is also scalable, new nodes can be added to the broker and thus increasing the message per second ratio that can be handled by the system. In this way an overall scalable architecture for message delivery is defined which can be extended with minimal administrative effort.

MINT Processing Interface is not limited in a certain type of processing or data schema that is delivered using the messages. This is achieved by utilizing a software design pattern named the Strategy or Policy Pattern, using this pattern it is possible to select different algorithms for execution on runtime.

The UML diagram that represents the Strategy or Policy pattern is depicted in Figure 3. Another benefit from using this design pattern is the abstraction that is introduced between the messaging layer of the system and the implementation of various algorithms for

processing. A developer does not have to know about the intrinsic details of the messaging system in order to implement another algorithm for data processing by Mint PI.

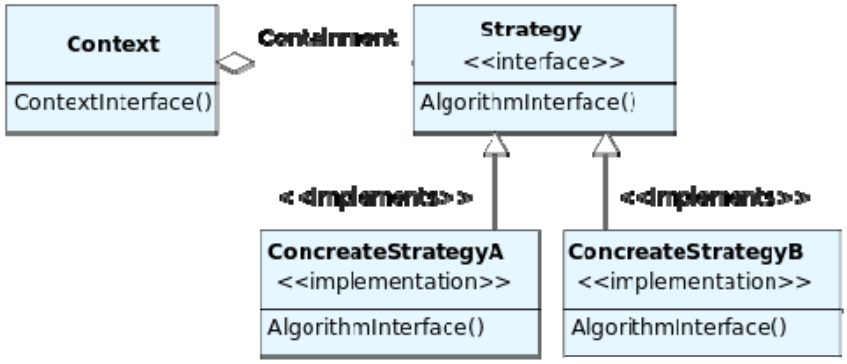


Figure 3: The Strategy or Policy Design Pattern (UML)

By following the architectural principles of Mint PI, a number of strategies have been implemented in order to support the delivery of EUscreen metadata records to Europeana. More specifically, the following strategies have been developed in order to support the functionalities defined by the Semantic Search Pilot of the Indicate Project:

1. A strategy for applying the EUscreen Schema to ESE V3.4 Schema XSLT to the metadata records.
2. A Strategy for enriching the records with multilingual subjects using the EUscreen thesauri.
3. A strategy for checking the various URLs that are included inside the metadata records for their validity.
4. Finally, a strategy that stores the records in the appropriate format inside the OAI-PMH repository that is used as an interoperability API between EUscreen and Europeana.

Initially, the producer is iterating the EUscreen metadata records directly from the Lucene index that is used by the EUscreen portal. In this way it is possible to filter the records based on the data provider and their status, e.g. if they are visible or not. These records are encapsulated inside a message in JSON serialization including the appropriate XSLT to be applied and various instructions for the complete set of strategies that will be applied on each one.

Each worker thread fetches one message from the queue and applies in a predefined sequence the implemented strategies. The first step is to apply the XSLT and generate the ESEV3.4 record, the next step is to enrich the records with multilingual thesauri terms for the case of the dc:subjects elements using the EUscreen Thesauri. When the transformation and enrichment is done, the final steps are to check the resulting records regarding the accessibility of the digital resources, e.g. thumbnails, and finally to store the records inside the OAI-PMH repository and thus making them accessible for harvesting by Europeana.

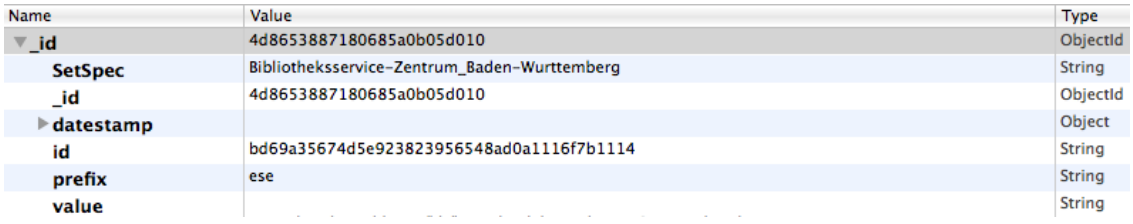
A core characteristic of the Ingestion Tool is that of being agnostic in regard to the Schema of an imported dataset, a characteristic that is also inherited to the EUscreen OAI-PMH platform. For this to be achieved the data layer of the platform has to be able to handle

heterogeneous metadata Schemata. Based on this assumption, it was decided to implement the underlying data layer using a NoSQL solution that does not enforce any particular schema and thus it will be able to adapt to metadata records that conform to different Schemata.
The NoSQL solution that was used for the EUscreen OAI-PMH Platform is the MongoDB document database. MongoDB is designed around the concept of documents that are internally implemented as JSON document and internally stored using the BSON format which stands for a binary serialization of the JSON format. It allows the existence of JSON documents in the same database or even collection having different fields and thus it does not enforce any specific data model schema. Finally it provides a rich set of native implementations of drivers for communicating with the database while the JSON format provides added value in the development of web application because the stored data do not have to be transformed to a different format in order to be consumed by the applications.

The EUscreen OAI-PMH Platform Data Layer is designed around three distinct collections that exist inside a MongoDB data base, collections can be perceived as tables of typical SQL databases, although it is not required that each document conforms to a specific datamodel and set of fields. These collections are the following:

1. Registry: in this collection the actual metadata records are stored and accessed by the implemented OAI-PMH verbs.
2. Conflicts: every time a new dataset is imported in the OAI-PMH repository, it is checked for the existence of duplicate records, if any exist, they are reported and stored in a different collection while they are also associated with a specific Report document.
3. Reports: every time an operation occurs on the OAI-PMH Repository, a report is created which includes any useful information regarding the operation. For example, in the case of import if any conflict is identified between the items it is reported for further reference.

The Registry collection constitutes the core collection of documents for the OAI-PMH repository. It contains all the records that it is desired to be exposed via the OAI-PMH repositories. Each record is stored inside a JSON document which also contains various other information that might be useful to the platform. More specifically, the record document contains information regarding the organization to which the item belongs to, a unique hash key that is generated by calculating the SHA1 hash of the string representation of the item, a datestamp which represents the date and time the record was inserted and finally a namespace "prefix" value which is required by the OAI-PMH specification. An example of a Registry Document instance is depicted in Figure 4.



Name	Value	Type
▼ _id	4d8653887180685a0b05d010	ObjectId
SetSpec	Bibliotheksservice-Zentrum_Baden-Wurttemberg	String
_id	4d8653887180685a0b05d010	ObjectId
▶ datestamp		Object
id	bd69a35674d5e923823956548ad0a1116f7b1114	String
prefix	ese	String
value		String

Figure 4: the structure of the Registry Document.

As it was mentioned earlier, another important collection of documents that is stored as part of the OAI-PMH repository is the Reports collection. The documents that are stored in this collection represent a set of valuable information that corresponds to specific actions of the repository platform. These actions are stored as values in the type attribute of the document and take one of the following values:

1. Add: this type represent an addition action in which records are added in the registry.
2. Update: this type represents an update action in which a set for a specific import already exist and it is updated by adding new metadata records.
3. Delete: this type represents the action of deleting a number of records from a specific import that already exists in the registry.

Apart from the action type a number of other values are also stored as part of the Report Document. More specifically, a set of valuable statics are stored; the number of conflicted items that were identified, the total number of the inserted records and the total number of items which corresponds to sum of the inserted records plus the conflicts. Two datestamps are also stored as part of the Report document, one datestamp corresponds to the time of creation of the document and the other to the time of closing of the document, in this way it is possible for the repository to calculate the time it took to import a whole data set into the database. Finally the date the import was published to the Ingestion Tool is stored together with the name of the organization it belongs to. A visual representation of the Report document structure is depicted in Figure 5.

ConflictsNumber	0	Int
InsertedNumber	9576	Int
TotalItems	9576	Int
_id	4cda3279100d685a312b47c8	ObjectId
▼closed		Object
date	2010/11/10	String
time	07:49:53	String
▼created		Object
date	2010/11/10	String
time	07:49:45	String
orgName	Rybinsk_State_History,_Architecture_and_Art_Museum-reserve	String
▼publicationDate		Object
date	2010-07-11	String
time	18:05:30.303	String
type	add	String

Figure 5: the structure of the Report document.

The last collection of the OAI-PMH repository database is the conflicts collection. The documents stored in this contain any metadata records that at the time of the exporting of an import from the Ingestion Tool to the OAI-PMH repository were found to be conflicted. These documents are quite simple in their structure. They contain an SHA1 hash of the conflicted item that was found together with the record and a reference to the Report document that it belongs to. In this way it is possible for someone to browse the actions that were made on the repository (e.g. additions, deletions and updates) and directly view the items that were found as conflicted for the cases of additions and updates. An example of a conflict document is depicted in Figure 6. It has to be noted that the whole procedure of creating unique hash codes and identifying conflicted items is an important functionality of the EUscreen OAI-PMH repository platform because it provides a mechanism for creating unique ids for the metadata records and also a mechanism for identifying duplicate.

▼ _id	4cda364c100d685a338d57c8	ObjectId
_id	4cda364c100d685a338d57c8	ObjectId
hash	74ec94f91b59e6d0e509d6615c776256dabb7647	String
orgName	Bildarchiv_Foto_Marburg	String
reportId		String

Figure 6: the structure of the Conflict document

On top of the data layer that was described so far a set of functionalities is built and more specifically an RSS Feed based on Atom and of course the implementation of the actual OAI-PMH verbs. The implementation of the verbs is based on the customization of the oaicat API which provides an abstract implementation of the OAI-PMH v2.0 specified verbs that can be customized in order to operate on top of different data layer technologies (e.g. flat XML files, relational databases etc.).

The verbs that were implemented in order to work with the current OAI-PMH Repository implementation are the following:

- Identify: this verb provides basic information regarding the running instance of the OAI-PMH v2.0 data repository such as, contact details of the admin of the repository, the base url that can be used by a harvester, a sample of an identifier among others. For a complete list of the information provided by the Identify verb someone can refer to the OAI-PMH specification. The information served by this verb does not have to be stored in the underlying data layer but is part of the configuration files of the verbs implementation.
- GetRecord: given the identifier of a record and the desired namespace prefix, this verb fetches from the MongoDB database the corresponsive Metadata record and delivers it as a response to the harvesting client. In the current implementation a query is executed based on the prefix which is part of the Registry Document and the unique id which is generated by the SHA1 hashing of the initial Metadata Record, this query corresponds to an exact match on the database.
- ListIdentifiers: given a set name and a namespace prefix, this verb responds with a list of identifiers of items that correspond to these criteria. The set name is identical to the name of the organization. In this way it is possible to organize the records of the repository around organizations/providers. Again, the namespace prefix is matched with the prefix field of the Registry Document.
- ListRecords: this verb operates in a similar way to the ListIdentifiers with the main difference being that instead of returning only the identifiers, the complete Metadata Record is served.
- ListMetadataFormats: this verb is implemented by aggregating all the unique prefix values that are stored in the data layer by executing an aggregation for uniqueness query on the Registry collection. The resulting response that is served by this verb contains all the unique namespace prefixes that exist in the OAI-PMH repository and can be used for accessing the Metadata Records.
- ListSets: this verb returns a list of all the sets that exist in the OAI-PMH repository. Sets are named after the organizations that provide metadata records to the OAI-PMH repository, in this way it is possible for someone to retrieve only the records that are associated with a specific organization. The way the values are extracted is similar to the ListMetadataFormats verb.

In every case that is needed, the verbs are implemented in such a way that they support paging through the mechanism of resumption tokens as it is defined by the OAI-PMH specification. The number of the returned items is specified through the configuration files of the oaicat running instance. Finally, by being a servlet implementation, the oaicat specific instance can be served through any of the available servlet contains that exist, e.g. tomcat, jboss, jetty etc. Currently it is served via a running Apache tomcat instance.

The RSS feed is implemented following the Atom Syndication Format which is an XML language used for web feeds while the Atom Publishing Protocol (APP) is a simple HTTP based protocol for creating and updating web resources. The purpose of an RSS feed in the current OAI-PMH repository implementation is to provide a mechanism for notifying metadata consumers for the occurrence of specific actions, for example when new items are added or updated to the repository in an automatic way. This is achieved by creating the RSS Feed on top of the Reports collections that was described earlier, in this way every time a new report is generated the feed is automatically updated and the subscribers are informed for the associated action. The implementation of the RSS Feed service is based on the Apache Abdera project.

The goal of the Apache Abdera project is to build a functionally-complete, high performance implementation of the IETF Atom Syndication Format (RFC 4287) and Atom Publishing Protocol (RFC 5023) specifications. The current implementation of the RSS Feed which is based on the Apache Abdera project, is built by communicating directly with the MongoDB based data layer of the OAI-PMH repository, every time a new Report document is inserted, it is also transformed into the Atom protocol XML representation and published on the Atom Feed that is maintained through the API of Apache Abdera. Finally, an Apache Abdera instance can be served via any of the available servlet containers, but in the current implementation it is contained in a Jetty servlet container for performance reasons.

3 Mapping Table

The following Table illustrate the mappings from the Euscreen elements to the Europeana Semantic Elements (ESE).

EUscreen element	ESE Element	ESE label
Identifier	dc:identifier	Identifier
Material Type	dc:type	Type
Title	dcterms:alternative	Title
Title in English	dc:title@xml:lang="en"	Title
Series Title	dcterms:isPartOf	Is Part Of
Series Title in English	dcterms:isPartOf@xml:lang="en"	Is Part Of
Clip Title	dcterms:alternative	Alternative Title
Summary	dc:description	Description
Summary in English	dc:description@xml:lang="en"	Description
Extended description	europeana:unstored	Unstored
Local keywords	europeana:unstored	Unstored
Thesaurus terms	dc:subject@xml:lang="en"	Subject
Geographical Coverage	dcterms:spatial@xml:lang="en"	Spatial Coverage
Genre	dc:type	Type
Topic	dc:subject@xml:lang="en"	Subject
Provider	europeana:dataProvider	Europeana Data Provider
Publisher/Broadcaster	dc:publisher	Publisher
First Broadcast channel	europeana:unstored	Unstored
Broadcast date	dcterms:issued	Date issued
Production year	dc:date	Date
Country of production	n.a.	-
Original identifier	europeana:unstored	Unstored
IPR restrictions	europeana:rights	Europeana Rights
Rights terms and conditions	dc:rights	Rights
Item type	n.a.	-
Item duration	dcterms:extent	Extent
Item colour	dcterms:medium	Medium
Item sound	dcterms:medium	Medium
Aspect ratio	dcterms:medium	Medium
Language used	dc:language	Language
Subtitle Language	n.a.	-
Original language	dc:language	Language
Contributor	dc:contributor	Contributor
Information	europeana:unstored	Unstored
Metadata language	n.a.	-
URI	europeana:isShownAt	isShownAt
Relation	n.a.	-
Relation type	n.a.	-
File name	n.a.	-
*value associated with the provider**	europeana:country	Country
"EUscreen"*	europeana:provider	Europeana Provider

*these values are constants

Open issues regarding the mapping towards ESE

The EUscreen metadata schema has some specific characteristics that make it particularly rich in information compared to ESE. The underlying use of a multilingual thesaurus for the classification of the items' subject and the systematic translation in English of the titles and the items' descriptions, have made the mapping towards the ESE v3.4 schema not always complete, leaving out sometimes some of the information present in the EUscreen schema. In particular, we were not able to fully map into ESE the EUscreen multilingual thesaurus terms used to classify our items in 12 different languages. Actually, in fact, in the Europeana portal a proper use of the xml:lang attribute is still (August 2012) not supported, and the dc:subject element in ESE, that we used to map our thesaurus terms, cannot be displayed and

filtered based on the value of the xml:lang attribute. So, to grant a proper display in the Europeana portal of this element, we had to map and export to ESE only the thesaurus terms in English (with the attribute xml:lang=”en”), leaving out all the other languages.
We faced a similar problem in the mapping of the “Summary”, “Summary in English”, “Extended description” and “Information”. These are different elements in the EUscreen schema that semantically can be mapped only to one element in ESE, the “dc:description”.

Unfortunately we were not able to distinguish in some way these different elements inside the “dc:description” container, neither to give them a fixed order of display. And for this reason, to grant again a proper display of these elements in the Europeana portal, we were forced to map and export only the “Summary” and the “Summary in English” in the “dc:description” element, mapping instead the “Extended description” and the “Information” elements in the “europeana:unstored” element, making them in this way not visible in the Europeana portal.

The consortium (notably the technical partners at NTUA) is in direct contact with the ingestion team at Europeana regarding the issues above. This input will be taken into account in future updates of the metadata standards maintained as by Europeana.

4 Appendix A: Euscreen-->ESE

```
<?xml version="1.0" encoding="UTF-8"?>

<!--

This file was generated by Altova MapForce 2009sp1



YOU SHOULD NOT MODIFY THIS FILE, BECAUSE IT WILL BE

OVERWRITTEN WHEN YOU RE-RUN CODE GENERATION.



Refer to the Altova MapForce Documentation for further details.

http://www.altova.com/mapforce

-->

<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:dcmitype="http://purl.org/dc/dcmitype/"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:europeana="http://www.europeana.eu/schemas/ese/"
xmlns:europeana2="http://www.euscreen.eu/schemas/euscreen/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:vmf="http://www.altova.com/MapForce/UDF/vmf"
xmlns:fn="http://www.w3.org/2005/xpath-functions" xmlns:grp="http://www.altova.com/Mapforce/grouping"
exclude-result-prefixes="europeana2 fn grp vmf xs xsi xsl">

	<xsl:template name="vmf:vmf1_inputtoresult">

		<xsl:param name="input" select="()"/>

		<xsl:choose>

			<xsl:when test="$input='Still'">

				<xsl:value-of select="'IMAGE'"/>

			</xsl:when>

			<xsl:when test="$input='Document'">

				<xsl:value-of select="'TEXT'"/>

			</xsl:when>

			<xsl:when test="$input='Audio'">

				<xsl:value-of select="'SOUND'"/>

			</xsl:when>

			<xsl:when test="$input='Video'">
```

```
                    <xsl:value-of select="'VIDEO'"/>
                </xsl:when>
            </xsl:choose>
        </xsl:template>
        <xsl:template name="vmf:vmf2_inputtoresult">
            <xsl:param name="input" select="()"/>
            <xsl:choose>
                <xsl:when test="$input='Rights Reserved - Free Access'">
                    <xsl:value-of select="'http://www.europeana.eu/rights/rr-f/'"/>
                </xsl:when>
                <xsl:when test="$input='Rights Reserved - Paid Access'">
                    <xsl:value-of select="'http://www.europeana.eu/rights/rr-p/'"/>
                </xsl:when>
                <xsl:when test="$input='Rights Reserved - Restricted Access'">
                    <xsl:value-of select="'http://www.europeana.eu/rights/rr-r/'"/>
                </xsl:when>
                <xsl:when test="$input='Unknown'">
                    <xsl:value-of select="'http://www.europeana.eu/rights/unknown/'"/>
                </xsl:when>
            </xsl:choose>
        </xsl:template>
        <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
        <xsl:template match="/">
            <europeana:metadata>
                <xsl:attribute name="xsi:schemaLocation" separator=" ">
                    <xsl:sequence select="'http://www.europeana.eu/schemas/ese/
C:/Users/cpard/Desktop/Projects/euscreen/ESEV3.4.xsd'"/>
                </xsl:attribute>
                <xsl:variable name="var1_instance" as="node()" select="."/>
                <xsl:for-each select="$var1_instance/europeana2:EUScreen/europeana2:metadata">
```

```
                                <xsl:variable name="var2_metadata" as="node()" select="."/>

                                <europeana:record>

                                        <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:TitleSet/europeana2:TitleSetInEnglish/europeana
2:title">

                                                <dc:title>

                                                        <xsl:attribute name="xml:lang">

                                                                <xsl:sequence
select="xs:string(xs:string('en'))"/>

                                                        </xsl:attribute>

                                                        <xsl:sequence select="xs:string(.)"/>

                                                </dc:title>

                                        </xsl:for-each>

                                        <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:TitleSet">

                                                <xsl:variable name="var6_TitleSet" as="node()"
select="."/>

                                                <xsl:for-each
select="europeana2:TitleSetInOriginalLanguage/europeana2:title">

                                                        <xsl:variable name="var8_title" as="node()"
select="."/>

                                                        <xsl:for-each
select="$var6_TitleSet/europeana2:TitleSetInEnglish/europeana2:title">

                                                                <xsl:if test="fn:exists((if
((xs:string($var8_title) = xs:string(.))) then () else xs:string($var8_title)))">

                                                                        <dcterms:alternative>

                                                                                <xsl:sequence
select="(if ((xs:string($var8_title) = xs:string(.))) then () else xs:string($var8_title))"/>

                                                                        </dcterms:alternative>

                                                                </xsl:if>

                                                        </xsl:for-each>

                                                </xsl:for-each>

                                        </xsl:for-each>

                                        <xsl:for-each select="europeana2:ContentDescriptiveMetadata">
```

```
                                                <xsl:variable
name="var12_ContentDescriptiveMetadata" as="node()" select="."/>

                                                <xsl:for-each select="europeana2:clipTitle">

                                                        <xsl:variable name="var14_clipTitle"
as="node()" select="."/>

                                                <xsl:for-each
select="$var12_ContentDescriptiveMetadata/europeana2:TitleSet/europeana2:TitleSetInOriginalLanguage/eur
opeana2:title">

                                                                <xsl:if test="fn:exists((if
((xs:string($var14_clipTitle) = xs:string(.))) then () else xs:string($var14_clipTitle)))">

                                                                        <dcterms:alternative>

                                                                                <xsl:sequence
select="(if ((xs:string($var14_clipTitle) = xs:string(.))) then () else xs:string($var14_clipTitle))"/>

                                                                        </dcterms:alternative>

                                                                </xsl:if>

                                                        </xsl:for-each>

                                                </xsl:for-each>

                                        </xsl:for-each>

                                        <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:ThesaurusTerm">

                                                <dc:subject>

                                                        <xsl:attribute name="xml:lang">

                                                                <xsl:sequence
select="xs:string(xs:string('en'))"/>

                                                        </xsl:attribute>

                                                        <xsl:sequence select="xs:string(.)"/>

                                                </dc:subject>

                                        </xsl:for-each>

                                        <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:topic">

                                                <dc:subject>

                                                        <xsl:attribute name="xml:lang">

                                                                <xsl:sequence
select="xs:string(xs:string('en'))"/>

                                                        </xsl:attribute>
```

```
                                                <xsl:sequence select="xs:string(.)"/>
                                            </dc:subject>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:summary">
                                            <dc:description>
                                                <xsl:sequence select="xs:string(.)"/>
                                            </dc:description>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:summaryInEnglish">
                                            <dc:description>
                                                <xsl:attribute name="xml:lang">
                                                    <xsl:sequence
select="xs:string(xs:string('en'))"/>
                                                </xsl:attribute>
                                                <xsl:sequence select="xs:string(.)"/>
                                            </dc:description>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:AdministrativeMetadata/europeana2:publisherbroadcaster">
                                            <dc:publisher>
                                                <xsl:sequence select="xs:string(.)"/>
                                            </dc:publisher>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:contributor">
                                            <dc:contributor>
                                                <xsl:sequence select="xs:string(.)"/>
                                            </dc:contributor>
                                        </xsl:for-each>
                                        <xsl:for-each
```

```
select="europeana2:ObjectDescriptiveMetadata/europeana2:SpatioTemporalInformation/europeana2:Temporal
Information/europeana2:productionYear">
                                                <dc:date>
                                                        <xsl:sequence select="xs:string(.)"/>
                                                </dc:date>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:SpatioTemporalInformation/europeana2:Temporal
Information/europeana2:broadcastDate">
                                                <dcterms:issued>
                                                        <xsl:sequence select="xs:string(.)"/>
                                                </dcterms:issued>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:TechnicalInformation/europeana2:materialType">
                                                <dc:type>
                                                        <xsl:sequence select="xs:string(.)"/>
                                                </dc:type>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:genre">
                                                <dc:type>
                                                        <xsl:attribute name="xml:lang">
                                                                <xsl:sequence
select="xs:string(xs:string('en'))"/>
                                                        </xsl:attribute>
                                                        <xsl:sequence select="xs:string(.)"/>
                                                </dc:type>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:TechnicalInformation/europeana2:itemDuration">
                                                <xsl:if test="fn:exists((if ((xs:string(.) != '00:00:00')) then
xs:string(.) else ()))">
                                                        <dcterms:extent>
```

```
                                                    <xsl:sequence select="(if ((xs:string(.)
!= '00:00:00')) then xs:string(.) else ())"/>
                                                </dcterms:extent>
                                            </xsl:if>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:TechnicalInformation/europeana2:aspectRatio">
                                            <dcterms:medium>
                                                <xsl:sequence select="xs:string(.)"/>
                                            </dcterms:medium>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:TechnicalInformation/europeana2:itemColor">
                                            <dcterms:medium>
                                                <xsl:sequence select="xs:string(.)"/>
                                            </dcterms:medium>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:TechnicalInformation/europeana2:itemSound">
                                            <dcterms:medium>
                                                <xsl:sequence select="xs:string(.)"/>
                                            </dcterms:medium>
                                        </xsl:for-each>
                                        <xsl:for-each
select="europeana2:AdministrativeMetadata/europeana2:identifier">
                                            <dc:identifier>
                                                <xsl:sequence select="xs:string(.)"/>
                                            </dc:identifier>
                                        </xsl:for-each>
                                        <xsl:variable name="var48_cond_result_exists" as="xs:string?">
                                            <xsl:choose>
                                                <xsl:when
```

```
test="fn:exists(europeana2:ObjectDescriptiveMetadata/europeana2:LanguageInformation/europeana2:language
Used)">

                                                                    <xsl:variable
name="var53_map_select_ObjectDescriptiveMetadata" as="xs:string*">

                                                                    <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:LanguageInformation/europeana2:languageUsed"
>

                                                                        <xsl:sequence
select="xs:string(.)"/>

                                                                    </xsl:for-each>

                                                                </xsl:variable>

                                                                <xsl:if
test="fn:exists($var53_map_select_ObjectDescriptiveMetadata)">

                                                                    <xsl:sequence
select="xs:string(fn:string-join($var53_map_select_ObjectDescriptiveMetadata, ' '))"/>

                                                                </xsl:if>

                                                            </xsl:when>

                                                            <xsl:otherwise>

                                                                <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:originallanguage">

                                                                    <xsl:sequence
select="xs:string(.)"/>

                                                                </xsl:for-each>

                                                            </xsl:otherwise>

                                                        </xsl:choose>

                                                    </xsl:variable>

                                                    <xsl:for-each select="$var48_cond_result_exists">

                                                        <dc:language>

                                                            <xsl:sequence select="."/>

                                                        </dc:language>

                                                    </xsl:for-each>

                                                    <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:TitleSet">

                                                        <xsl:variable name="var58_TitleSet" as="node()"
select="."/>
```

```
                                                <xsl:for-each
select="europeana2:TitleSetInOriginalLanguage/europeana2:seriesTitle">

                                                    <xsl:variable name="var60_seriesTitle"
as="node()" select="."/>

                                                    <xsl:for-each
select="$var58_TitleSet/europeana2:TitleSetInEnglish/europeana2:seriesTitle">

                                                        <xsl:if test="fn:exists((if
((xs:string($var60_seriesTitle) = xs:string(.))) then () else xs:string($var60_seriesTitle)))">

                                                            <dcterms:isPartOf>

                                                                <xsl:sequence
select="(if ((xs:string($var60_seriesTitle) = xs:string(.))) then () else xs:string($var60_seriesTitle))"/>

                                                            </dcterms:isPartOf>

                                                        </xsl:if>

                                                    </xsl:for-each>

                                                </xsl:for-each>

                                                <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:TitleSet/europeana2:TitleSetInEnglish/europeana
2:seriesTitle">

                                                    <dcterms:isPartOf>

                                                        <xsl:attribute name="xml:lang">

                                                            <xsl:sequence
select="xs:string(xs:string('en'))"/>

                                                        </xsl:attribute>

                                                        <xsl:sequence select="xs:string(.)"/>

                                                    </dcterms:isPartOf>

                                                </xsl:for-each>

                                                <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:SpatioTemporalInformation/europeana2:SpatialInf
ormation/europeana2:GeographicalCoverage">

                                                    <dcterms:spatial>

                                                        <xsl:attribute name="xml:lang">

                                                            <xsl:sequence
select="xs:string(xs:string('en'))"/>

                                                        </xsl:attribute>
```

```
                                        <xsl:sequence select="xs:string(.)"/>
                                    </dcterms:spatial>
                                </xsl:for-each>
                                <xsl:for-each
select="europeana2:AdministrativeMetadata/europeana2:rightsTermsAndConditions">
                                    <dc:rights>
                                        <xsl:sequence select="xs:string(.)"/>
                                    </dc:rights>
                                </xsl:for-each>
                                <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:extendedDescription">
                                    <europeana:unstored>
                                        <xsl:sequence select="xs:string(.)"/>
                                    </europeana:unstored>
                                </xsl:for-each>
                                <xsl:for-each
select="europeana2:ContentDescriptiveMetadata/europeana2:LocalKeyword">
                                    <europeana:unstored>
                                        <xsl:sequence select="xs:string(.)"/>
                                    </europeana:unstored>
                                </xsl:for-each>
                                <xsl:for-each
select="europeana2:AdministrativeMetadata/europeana2:firstBroadcastChannel">
                                    <europeana:unstored>
                                        <xsl:sequence select="xs:string(.)"/>
                                    </europeana:unstored>
                                </xsl:for-each>
                                <xsl:for-each
select="europeana2:AdministrativeMetadata/europeana2:originalIdentifier">
                                    <europeana:unstored>
                                        <xsl:sequence select="xs:string(.)"/>
                                    </europeana:unstored>
```

```
                                        </xsl:for-each>

                                        <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:information">

                                                <europeana:unstored>

                                                        <xsl:sequence select="xs:string(.)"/>

                                                </europeana:unstored>

                                        </xsl:for-each>

                                        <xsl:for-each
select="europeana2:AdministrativeMetadata/europeana2:identifier">

                                                <europeana:object>

                                                        <xsl:sequence
select="xs:string(xs:anyURI(fn:concat(fn:concat(fn:concat(fn:concat(fn:concat(fn:concat('http://images1.noterik.com/domain/euscreen/thumbs/', fn:substring(fn:substring-after(xs:string(.), 'EUS_'), xs:double(xs:decimal(1)), xs:double(xs:decimal(1)))), '/'), fn:substring(fn:substring-after(xs:string(.), 'EUS_'), xs:double(xs:decimal(2)), xs:double(xs:decimal(1)))), '/'), xs:string(.)), '.jpg')))"/>

                                                </europeana:object>

                                        </xsl:for-each>

                                        <europeana:provider>

                                                <xsl:sequence select="'EUscreen Project'"/>

                                        </europeana:provider>

                                        <xsl:for-each
select="europeana2:ObjectDescriptiveMetadata/europeana2:TechnicalInformation/europeana2:materialType">

                                                <xsl:variable name="var84_result_vmf1_inputtoresult"
as="xs:string?">

                                                        <xsl:call-template
name="vmf:vmf1_inputtoresult">

                                                                <xsl:with-param name="input"
select="xs:string(.)"/>

                                                        </xsl:call-template>

                                                </xsl:variable>

                                                <xsl:if
test="fn:exists($var84_result_vmf1_inputtoresult)">

                                                        <europeana:type>

                                                                <xsl:sequence
select="$var84_result_vmf1_inputtoresult"/>

                                                        </europeana:type>
```

```
                                        </xsl:if>
                                    </xsl:for-each>
                                    <xsl:for-each
select="europeana2:AdministrativeMetadata/europeana2:iprRestrictions">
                                        <xsl:variable name="var87_result_vmf2_inputtoresult"
as="xs:string?">
                                            <xsl:call-template
name="vmf:vmf2_inputtoresult">
                                                <xsl:with-param name="input"
select="xs:string(.)"/>
                                            </xsl:call-template>
                                        </xsl:variable>
                                        <xsl:if
test="fn:exists($var87_result_vmf2_inputtoresult)">
                                            <europeana:rights>
                                                <xsl:sequence
select="xs:string(xs:anyURI($var87_result_vmf2_inputtoresult))"/>
                                            </europeana:rights>
                                        </xsl:if>
                                    </xsl:for-each>
                                    <xsl:for-each
select="europeana2:AdministrativeMetadata/europeana2:provider">
                                        <europeana:dataProvider>
                                            <xsl:sequence select="xs:string(.)"/>
                                        </europeana:dataProvider>
                                    </xsl:for-each>
                                    <xsl:for-each
select="europeana2:AdministrativeMetadata/europeana2:identifier">
                                        <europeana:isShownAt>
                                            <xsl:sequence
select="xs:string(xs:anyURI(fn:concat('http://www.euscreen.eu/play.html?id=', xs:string(.))))"/>
                                        </europeana:isShownAt>
                                    </xsl:for-each>
                                </europeana:record>
```

```
                    </xsl:for-each>
                </europeana:metadata>
            </xsl:template>
</xsl:stylesheet>
```