Using Mapping Memory Manager (3M) with CIDOC CRM

Dominic Oldman
(DOldman@thebritishmuseum.ac.uk)
Maria Theodoridou
(maria@ics.forth.gr)
Georgios Samaritakis
(samarita@ics.forth.gr)

Draft
Version 4g

Contents

1 Introduction

1.1 What is 3M?

The 3M web application is an open source schema mapping tool developed by a consortium through an ongoing project called CultureBroker. A public implementation of 3M is available from the FORTH web site http://www.ics.forth.gr/isl/3M/. FORTH (Foundation for Research & Technology – Hellas) is the main developer of 3M and is also the administrative home of the CIDOC CRM (Conceptual Reference Model). Delving BV (www.delving.eu) is a software development company located in Holland and has contributed to the development of the X3ML engine, that handles the URI generation and the data transformation to rdf.

3M allows data experts to transform their internal structured data and other associated contextual knowledge to other schemas and, in particular, the CIDOC CRM (Conceptual Reference Model). Fields or elements from a source database (Source Nodes) are aligned with one or more entities described in the target schema so that the data from an entire system can be transformed. The purpose of this is typically for publication on the Web and in particular meaningful integration with other data also transformed to the same target schema.

Although 3M can map data to other schemas it is particularly aimed at mapping to contextual data and to CIDOC CRM, and alignment to it is based, not just on Source Nodes, but on other implicit information known to the owning organisation. Mapping data requires good communication with the producers of data because mapping, particularly to the CIDOC CRM, is not simply a technical exercise but provides the opportunity for additional knowledge to be encoded and published. The development of 3M attempts to support the need to involve and record the knowledge of the data producers and ensure the best possible semantic and contextual mapping of cultural data.

3M is part of an ongoing development project based on a Data Provisioning reference model called Synergy (http://www.cidoc-crm.org/docs/SRM_v0.1.pdf). Synergy describes the relationships and processes required for high quality data provisioning and data integration. It identifies the different processes and roles involved in data provisioning and data aggregation. 3M supports some of these processes but in particular the schema matching and X3ML transformation, including support for Linked Data Publication.

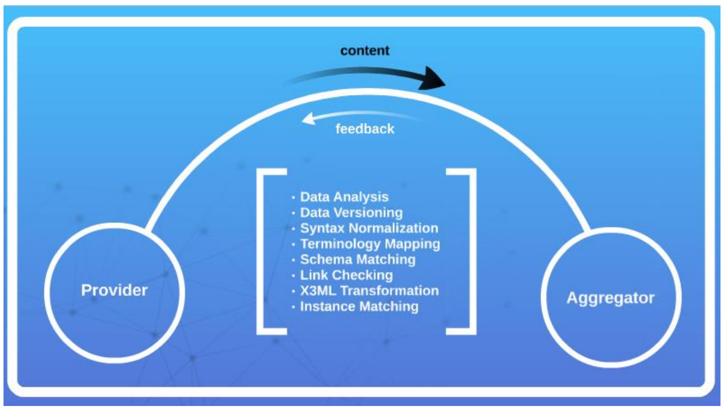


Figure 1: https://prezi.com/36wspw0bed6v/provider-and-aggregator/ (Gerald de Jong)

The aim of Synergy is to identify the different roles and processes which should be separated and which are currently combined. 3M takes these principles and attempts to separate out technical tasks from data representation tasks. In other mapping applications functions can be highly sophisticated but because they lack this separation they make it difficult for data experts to use them, and therefore mapping can remained locked into technical processes and roles.

1.2 Who should be Mapping Data?

Figure 2: Tim Clark, Expert and Head of Japanese, Dept. of Asia, British Museum

Contextual data mapping from internal (closed world) institutional systems is an initial step to developing a richer digital representation of cultural things. Much of the data produced by cultural heritage organisations describes objects in terms of their physical and technical properties. While this is useful information within institutions it is less useful to wider groups and audiences. Less attention is given to historical significance and relevance which is left to handcrafted textual narratives in other forms including books, exhibitions and e-publishing. Cultural Heritage cataloguing standards don’t support this broader information, yet digital audiences, like physical visitors, are likely to find this information more engaging. Contextual mapping is also crucial for research and provides the foundations necessary for this wider range of knowledge and information to be layered onto canonical institutional data. Data mapping should be seen as an initial part of a larger ecosystem for developing and integrating knowledge.

Traditionally mapping data has been seen as a mechanical process associated with information technologists and software developers. However, the skills required for mapping cultural heritage data to a contextual representation are not primarily information technology skills, and sit more comfortably with people who understand the data rather than people who understand the technology – although a collaboration is ideal!

Curators, for example, are skilled in disseminating information about objects to a wide range of audiences. When curators design an exhibition they create an environment in which relationships can be understood by many different types of visitor, often supporting particular historical narratives. Objects are not just lined up in broad categories with a label that lists a selection of their physical properties. Curators create an environment designed to answer questions about the relevance of objects and associated events in history, and their significance within the context of the exhibition. Ideally, an exhibition should inspire visitors and encourage them to make their own connections exploring their own knowledge and experience. Contextual data provides a means for people to explore a digital world in which curators and scholars have embedded the same type of knowledge used in exhibitions, but to establish relationships across a wider range of sources and across different historical narratives.

Technologists are familiar with processing information and integrating data between information systems as part of systems integration projects. Representing data semantically is a different task and semantic data mapping relies on a correspondence with the producers of information, not just an abstract database of rows and columns. For contextual data transformation, the bigger the distance between the process of mapping data and the collection/object experts, the more likely the outcome will be unsatisfactory and contain errors. As such curatorial staff and other experts who understand the objects the most, are the ideal data mappers and developers of digital knowledge representation.

1.3 Mapping Mentality

As already stated the correspondence between institutional knowledge and digital representation is not a narrow, field to field, schema to schema relationship. Source information is transformed into a semantic framework which is informed not just from the source data, but from unrecorded institutional knowledge. The more context and meaning that a mapping can incorporate the richer it will become allowing the internal database schema, based on artificial constructs, to be transformed into more universal 'real' concepts. Traditionally when we create databases we are thinking about internal organisational requirements and standard database nomenclature. We don't have to relate anything to real life concepts and it is sufficient that the organisation can, usually after some training and experience, understand what things mean and how to interact with the database. Explicit meaning is not applied to the database but is expanded by the software, user interfaces, training and manuals and the knowledge that organisational experts retain. Technologists are particularly used to working within these artificial boundaries and mould user concepts into a view of the data that they are more comfortable with.

Sarah Fagg - http://creativecommons.org/licenses/by-nc-nd/2.0/

The CIDOC CRM ontology is based on descriptions of reality (described scientifically) and there is no opportunity to, 'make it up' as you go along. We can't simply add another entity or field arbitrarily as we tend to do when adding new tables and fields to a database (and in the same way we can't invent a new animal, mineral, vegetable or invent and add another word to the dictionary). However, moving from an artificial view of data, in which people are working within the same organisational knowledge environment, to a real world representation of data, in which information is related to universal concepts, is something that requires a different approach and a reconditioning away from technology led practices. The so called 'complexity' of the CRM is simply a matter of aligning knowledge with real world general concepts – something that we are not used to doing and which only seems different because of the way in which data representation has been technology led. In reading this manual and other CIDOC CRM materials you are invited to take one of two pills. You can take a blue pill, in which case, "the story ends, you wake up in your bed and you believe whatever you want to believe", or you take the red pill and "see how far the rabbit hole goes". (The Matrix 1999)

2 CIDOC CRM

The CIDOC CRM is an ontology that establishes a set of universal concepts that are bound to reality. This eliminates the problem of integrating data in open data environments. Instead of attempting to integrate data based on artificial data profiles and terminological concepts, a practically impossible task, the CIDOC CRM harmonises data through a 'framework' of semantic relationships and entities (classes). This is very important because instead of simply producing unified models of fixed fields and values such as those in traditional union catalogues, the CRM allows data to retain its relationships and vocabularies (enhancing them semantically) so that these can be compared and studied rather than homogenised – something crucial to both scholarship and interesting engagement.

"Concepts are the units of thought — ideas, meanings, or (categories of) objects and events—which underlie many knowledge organization systems. As such, concepts exist in the mind as abstract entities which are independent of the terms used to label them."[1]

As such classification concepts provide poor support for integrating heterogeneous cultural heritage data and by themselves, without context, are not that useful.

The principles of the CIDOC CRM are documented in the CIDOC CRM Primer[2] and are summarised as follows.

1. The CIDOC CRM consists of a set of general *real world things* that can be connected through the use of properties or *relationships*.

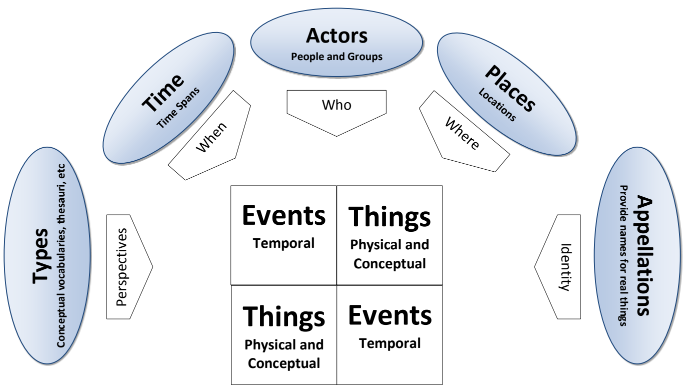


Figure 3: CIDOC CRM - High Level View

2. CIDOC CRM things and relationships have been abstracted as generalisations from large numbers of cultural heritage organisations and involved talking to a significant numbers of cultural heritage experts.
3. There are two key branches of the CIDOC CRM. There are things that have a persistent identity that can, by their nature, survive events (physical things or ideas and concepts), and there are the temporal concepts (like events) that have a nature of happening rather than being.
4. Things and relationships exist in a hierarchy of meanings that provide different levels of generalisation (or specialisation depending on the way you look at it) but harmonisation can occur across these levels.

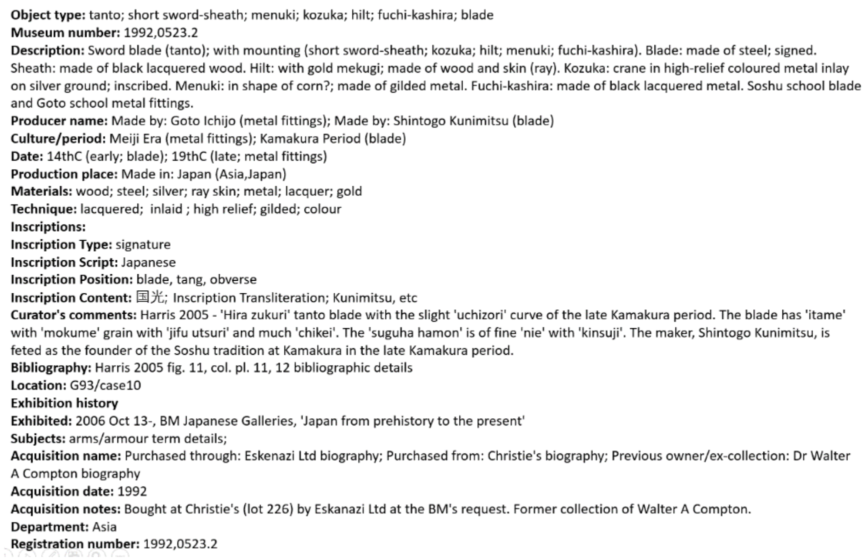
Here is a typical (informal) CRM statement:

An **object** (which has a type)
  -**was produced by**
    -an **event**
      -that **used** a particular **technique**
        (that is described by an authority term)
      -that **took place** in a particular time period
        (that is described by an authority term and date range)
      -that was **carried out by** a particular **person**
        (who is **identified** and described in a biographical record)
        (who is **identified** by one or more names)

[1] See W3C SKOS Primer
[2] www.cidoc-crm.org/docs/CRMPrimer_v1.1.pdf

3 Thinking in terms of Relationships & Events

We are used to describing objects with lists of labels and values. For example, this British Museum record:



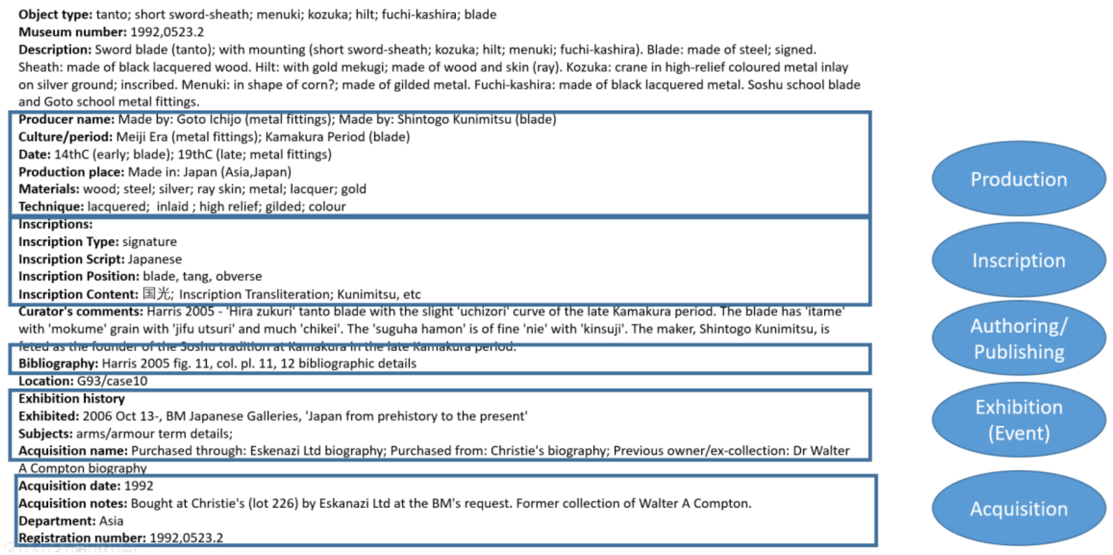
Object type: tanto; short sword-sheath; menuki; kozuka; hilt; fuchi-kashira; blade
Museum number: 1992,0523.2
Description: Sword blade (tanto); with mounting (short sword-sheath; kozuka; hilt; menuki; fuchi-kashira). Blade: made of steel; signed. Sheath: made of black lacquered wood. Hilt: with gold mekugi; made of wood and skin (ray). Kozuka: crane in high-relief coloured metal inlay on silver ground; inscribed. Menuki: in shape of corn?; made of gilded metal. Fuchi-kashira: made of black lacquered metal. Soshu school blade and Goto school metal fittings.
Producer name: Made by: Goto Ichijo (metal fittings); Made by: Shintogo Kunimitsu (blade)
Culture/period: Meiji Era (metal fittings); Kamakura Period (blade)
Date: 14thC (early; blade); 19thC (late; metal fittings)
Production place: Made in: Japan (Asia,Japan)
Materials: wood; steel; silver; ray skin; metal; lacquer; gold
Technique: lacquered; inlaid ; high relief; gilded; colour
Inscriptions:
Inscription Type: signature
Inscription Script: Japanese
Inscription Position: blade, tang, obverse
Inscription Content: 国光; Inscription Transliteration; Kunimitsu, etc
Curator's comments: Harris 2005 - 'Hira zukuri' tanto blade with the slight 'uchizori' curve of the late Kamakura period. The blade has 'itame' with 'mokume' grain with 'jifu utsuri' and much 'chikei'. The 'suguha hamon' is of fine 'nie' with 'kinsuji'. The maker, Shintogo Kunimitsu, is feted as the founder of the Soshu tradition at Kamakura in the late Kamakura period.
Bibliography: Harris 2005 fig. 11, col. pl. 11, 12 bibliographic details
Location: G93/case10
Exhibition history
Exhibited: 2006 Oct 13-, BM Japanese Galleries, 'Japan from prehistory to the present'
Subjects: arms/armour term details;
Acquisition name: Purchased through: Eskenazi Ltd biography; Purchased from: Christie's biography; Previous owner/ex-collection: Dr Walter A Compton biography
Acquisition date: 1992
Acquisition notes: Bought at Christie's (lot 226) by Eskanazi Ltd at the BM's request. Former collection of Walter A Compton.
Department: Asia
Registration number: 1992,0523.2

There are a few issues with this type of knowledge representation.

Firstly, it has little context and this can make the representation difficult to interpret beyond a simple level, both for humans and computers, with ambiguity particularly affecting integration with other data sources.

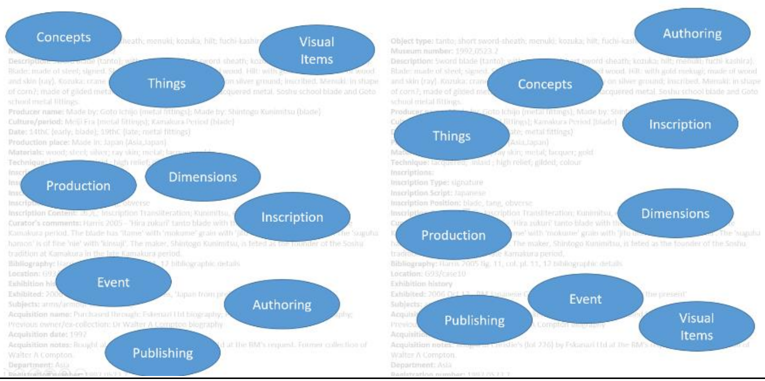
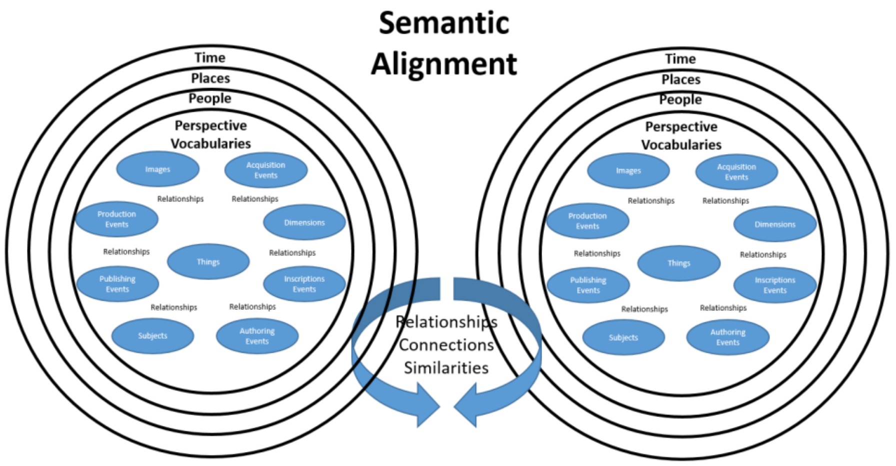
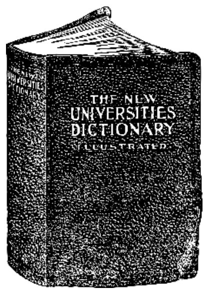
Secondly, regardless of the standards that may be employed by different organisations the exact meaning of fields and values can be different. For example, the term, 'knives' is interpreted and represented differently in different vocabularies, 'Creator' used by itself is highly generalised and used in different ways supporting a very wide range of meaning. The event and relationship form of CIDOC CRM avoids these issues but in doing so changes the way that you must think about the data and how it is represented.

The details of this object imply a range of different events or activities which are commonly understood by external observers. These are the events that provide a framework by which connections can be made with properties of context that also have universal application. Moreover, the full details of these event may not be fully known to the owner of the object and the event framework supports the addition of new facts without affecting the integrity of the rest of the data.



Object type: tanto; short sword-sheath; menuki; kozuka; hilt; fuchi-kashira; blade
Museum number: 1992,0523.2
Description: Sword blade (tanto); with mounting (short sword-sheath; kozuka; hilt; menuki; fuchi-kashira). Blade: made of steel; signed. Sheath: made of black lacquered wood. Hilt: with gold mekugi; made of wood and skin (ray). Kozuka: crane in high-relief coloured metal inlay on silver ground; inscribed. Menuki: in shape of corn?; made of gilded metal. Fuchi-kashira: made of black lacquered metal. Soshu school blade and Goto school metal fittings.
Producer name: Made by: Goto Ichijo (metal fittings); Made by: Shintogo Kunimitsu (blade)
Culture/period: Meiji Era (metal fittings); Kamakura Period (blade)
Date: 14thC (early; blade); 19thC (late; metal fittings)
Production place: Made in: Japan (Asia,Japan)
Materials: wood; steel; silver; ray skin; metal; lacquer; gold
Technique: lacquered; inlaid ; high relief; gilded; colour
Inscriptions:
Inscription Type: signature
Inscription Script: Japanese
Inscription Position: blade, tang, obverse
Inscription Content: 国光; Inscription Transliteration; Kunimitsu, etc
Curator's comments: Harris 2005 - 'Hira zukuri' tanto blade with the slight 'uchizori' curve of the late Kamakura period. The blade has 'itame' with 'mokume' grain with 'jifu utsuri' and much 'chikei'. The 'suguha hamon' is of fine 'nie' with 'kinsuji'. The maker, Shintogo Kunimitsu, is feted as the founder of the Soshu tradition at Kamakura in the late Kamakura period.
Bibliography: Harris 2005 fig. 11, col. pl. 11, 12 bibliographic details
Location: G93/case10
Exhibition history
Exhibited: 2006 Oct 13-, BM Japanese Galleries, 'Japan from prehistory to the present'
Subjects: arms/armour term details;
Acquisition name: Purchased through: Eskenazi Ltd biography; Purchased from: Christie's biography; Previous owner/ex-collection: Dr Walter A Compton biography
Acquisition date: 1992
Acquisition notes: Bought at Christie's (lot 226) by Eskanazi Ltd at the BM's request. Former collection of Walter A Compton.
Department: Asia
Registration number: 1992,0523.2

3.1 The Process of Semantic Harmonisation / Reconciliation

 Concepts Visual Items Things Production Dimensions Inscription Event Authoring Publishing Authoring Concepts Inscription Things Dimensions Production Event Publishing Visual Items	Different data records contain different information organised in different ways with different structures and vocabularies - but underlying this they contain common semantic concepts that are not explicitly stated or related.
 Semantic Alignment Time Places People Perspective Vocabularies Images Acquisition Events Relationships Production Events Dimensions Things Publishing Events Inscriptions Events Subjects Authoring Events Relationships Connections Similarities	By identifying the semantic framework and context implicit in the data, and making it explicit, these records can be harmonised while retaining the properties and details.
	This harmonised context provides a mechanism for matching instances of people, places and other entities.
	It provides an environment where different localised vocabularies can be compared as research objects.

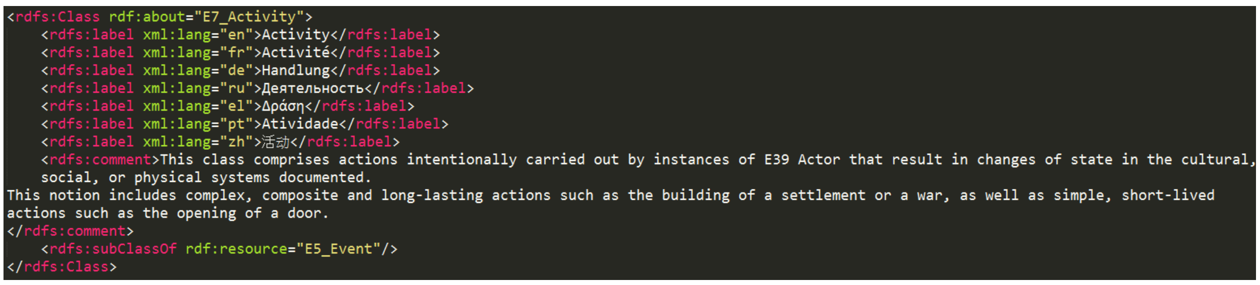
4 Resource Description Framework Schema (RDFS)

The CIDOC CRM is a knowledge representation system independent of any particular technology. When applying it to a technical implementation, like Linked Data, there will inevitably be some technical concepts that data mapping software use and that you should be aware of because they have some bearing on data mapping methods.

4.1 Classes and Properties

The CIDOC CRM is defined in a reference document[3] and there is an encoding available as a RDFS file for use in Linked Data projects. RDFS is the schema language for RDF which is the meta-model that defines triple statements, **Subject, Predicate & Object**. This is the model that supports data. RDFS is separate from the data but tells us something about it (starts to introduce an infrastructural level of semantics) and introduces the idea of inference and different levels of knowledge.

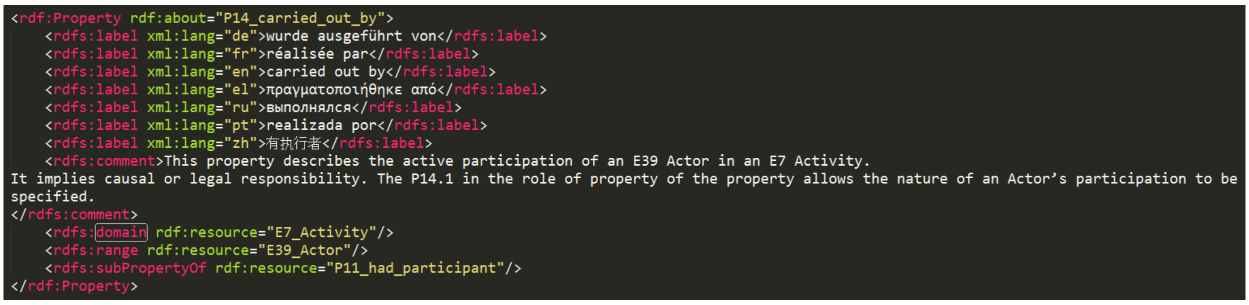
Every Entity in the CRM is defined as a Class – an RDFS Class. Here is a section of the CRM RDFS file. It is fairly simple. It defines a Class, in this case **E7_Activity**, some labels in different languages, a comment and then asserts that it is a Sub Class, or a specialisation, of **E5_Event.**



```
<rdfs:Class rdf:about="E7_Activity">
    <rdfs:label xml:lang="en">Activity</rdfs:label>
    <rdfs:label xml:lang="fr">Activité</rdfs:label>
    <rdfs:label xml:lang="de">Handlung</rdfs:label>
    <rdfs:label xml:lang="ru">Деятельность</rdfs:label>
    <rdfs:label xml:lang="el">Δράση</rdfs:label>
    <rdfs:label xml:lang="pt">Atividade</rdfs:label>
    <rdfs:label xml:lang="zh">活动</rdfs:label>
    <rdfs:comment>This class comprises actions intentionally carried out by instances of E39 Actor that result in changes of state in the cultural,
    social, or physical systems documented.
This notion includes complex, composite and long-lasting actions such as the building of a settlement or a war, as well as simple, short-lived
actions such as the opening of a door.
</rdfs:comment>
    <rdfs:subClassOf rdf:resource="E5_Event"/>
</rdfs:Class>
```

This means that **E7_Activity** is also a member of the class **E5_Event** and that when we talk about **Events,** this includes **Activities.** In the **CRM** there are also some more specific classes which are sub classes of **Activity** which have their own sub classes. These are simply specialisations. If I say that an **Activity** *was carried out by* an **Actor,** I could also say, if I knew the specifics, that **Production** *was carried out by* a **Person.** Through RDFS inference a query about what **Activities** had been *carried out by* an **Actor**, would retrieve **both** statements.

Equally, properties are defined in the RDFS file and these two can have sub properties. Here is the property definition for **P14_carried_out_by**



```
<rdf:Property rdf:about="P14_carried_out_by">
    <rdfs:label xml:lang="de">wurde ausgeführt von</rdfs:label>
    <rdfs:label xml:lang="fr">réalisée par</rdfs:label>
    <rdfs:label xml:lang="en">carried out by</rdfs:label>
    <rdfs:label xml:lang="el">πραγματοποιήθηκε από</rdfs:label>
    <rdfs:label xml:lang="ru">выполнялся</rdfs:label>
    <rdfs:label xml:lang="pt">realizada por</rdfs:label>
    <rdfs:label xml:lang="zh">有执行者</rdfs:label>
    <rdfs:comment>This property describes the active participation of an E39 Actor in an E7 Activity.
It implies causal or legal responsibility. The P14.1 in the role of property of the property allows the nature of an Actor’s participation to be
specified.
</rdfs:comment>
    <rdfs:domain rdf:resource="E7_Activity"/>
    <rdfs:range rdf:resource="E39_Actor"/>
    <rdfs:subPropertyOf rdf:resource="P11_had_participant"/>
</rdf:Property>
```

Notice that **carried_out_by** is a sub-property of **had_participant** which is a broader generalisation. When we use **P11_had_participant,** by inference we can automatically say that this also means, **P14_carried out by.** With most Linked Data systems we also have the choice of turning inference off.

4.2 Domain and Range

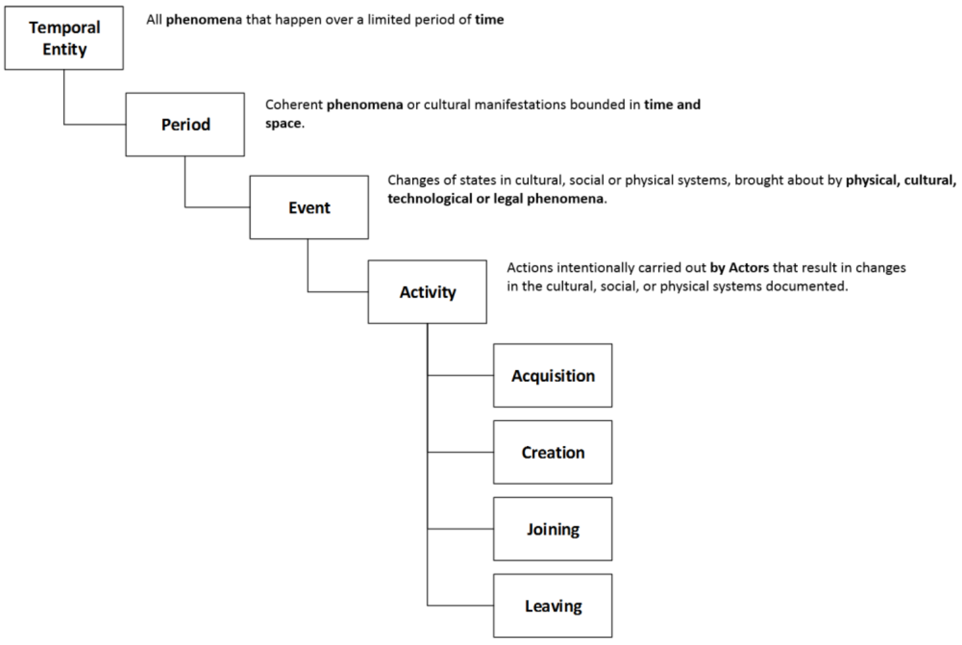
In the above encoding you can also see **rdfs:domain** and **rdfs:range**. This defines between what entities a property can be used. This is extremely important to maintain the integrity of the semantics within an ontology**. *P14_carried_out_by*** has a Domain of **E7_Activity** and a Range of **E39_Actor**. This means that its starting Domain can be **Activity** or (by inference) any sub class below it in the hierarchy of classes. Also that the entity that it ends with can be **E39_Actor** or

[3] http://www.cidoc-crm.org/official_release_cidoc.html

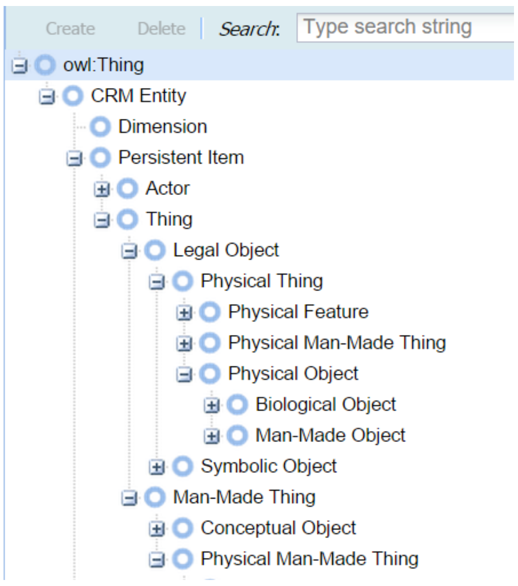
any sub classes below that. This is important for the mapping process and 3M uses the same terminology of Domain and Range.

5 Levels of Knowledge

The diagram below shows some of the hierarchy of the CIDOC CRM on a branch that shows **Temporal Entities**. We saw in the last section that entities are arranged as sub-classes of others. This provides levels of specialisation that allows different levels of knowledge to be applied. This also means that if, for example, alignment can only be made to the **Event** entity, that it will still integrate with data described as a particular **Activity.** All **Events** are **Temporal Entities** and all **Activities** are **Events**.


Temporal Entity
All phenomena that happen over a limited period of time
Period
Coherent phenomena or cultural manifestations bounded in time and space.
Event
Changes of states in cultural, social or physical systems, brought about by physical, cultural, technological or legal phenomena.
Activity
Actions intentionally carried out by Actors that result in changes in the cultural, social, or physical systems documented.
Acquisition
Creation
Joining
Leaving

A similar hierarchy is used with Persistent Items gradually moving from broad description of things that can be physical or conceptual and man-made or biological.



owl:Thing
CRM Entity
Dimension
Persistent Item
Actor
Thing
Legal Object
Physical Thing
Physical Feature
Physical Man-Made Thing
Physical Object
Biological Object
Man-Made Object
Symbolic Object
Man-Made Thing
Conceptual Object
Physical Man-Made Thing

Figure 4. Screenshot from Web Protege

Applications like Stanford's Web protégé allow you to browse the hierarchy and read the descriptions of the entities. (see http://webprotege.stanford.edu, and find the CIDOC CRM RDFS v6.0 entry in the listing). For example, a **Man-Made Thing** is described as;

"This class comprises discrete, identifiable man-made items that are documented as single units. These items are either intellectual products or man-made physical things, and are characterized by relative stability. They may for instance have a solid physical form, an electronic encoding, or they may be logical concepts or structures."

A **Physical Man-Made Thing** specialises to physical rather than immaterial things and the hierarchy branches into physical and conceptual. The description is;

"This class comprises all persistent physical items that are purposely created by human activity."

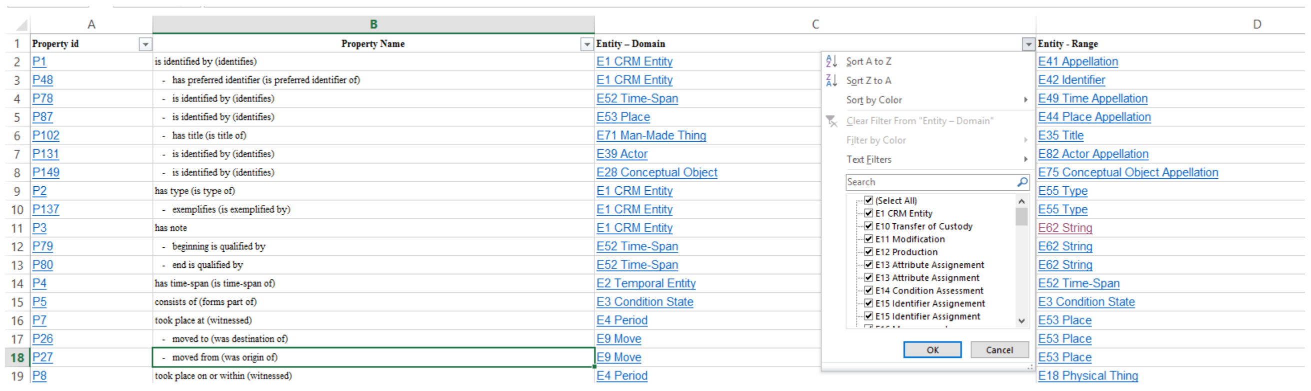
6 CIDOC CRM Domain/Range Table

The domain and range of CIDOC CRM relationships are produced in a table in the CIDOC CRM reference document. The following table shows some of these.

6.1 CIDOC CRM Property Hierarchy – sample:

Property id	Property Name	**Entity – Domain**	**Entity - Range**
P1	is identified by (identifies)	E1 CRM Entity	E41 Appellation
P48	- has preferred identifier (is preferred identifier of)	E1 CRM Entity	E42 Identifier
P78	- is identified by (identifies)	E52 Time-Span	E49 Time Appellation
P87	- is identified by (identifies)	E53 Place	E44 Place Appellation
P102	- has title (is title of)	E71 Man-Made Thing	E35 Title
P131	- is identified by (identifies)	E39 Actor	E82 Actor Appellation
P149	- is identified by (identifies)	E28 Conceptual Object	E75 Conceptual Object Appellation
P2	has type (is type of)	E1 CRM Entity	E55 Type
P137	- exemplifies (is exemplified by)	E1 CRM Entity	E55 Type
P3	has note	E1 CRM Entity	E62 String
P79	- beginning is qualified by	E52 Time-Span	E62 String
P80	- end is qualified by	E52 Time-Span	E62 String
P4	has time-span (is time-span of)	E2 Temporal Entity	E52 Time-Span
P5	consists of (forms part of)	E3 Condition State	E3 Condition State
P7	took place at (witnessed)	E4 Period	E53 Place
P26	- moved to (was destination of)	E9 Move	E53 Place
P27	- moved from (was origin of)	E9 Move	E53 Place
P8	took place on or within (witnessed)	E4 Period	E18 Physical Thing
P9	consists of (forms part of)	E4 Period	E4 Period
P10	falls within (contains)	E4 Period	E4 Period
P12	occurred in the presence of (was present at)	E5 Event	E77 Persistent Item
P111	- added (was added by)	E79 Part Addition	E18 Physical Thing
P113	- removed (was removed by)	E80 Part Removal	E18 Physical Thing
P11	- had participant (participated in)	E5 Event	E39 Actor
P14	- - carried out by (performed)	E7 Activity	E39 Actor
P22	- - - transferred title to (acquired title through)	E8 Acquisition	E39 Actor
P23	- - - transferred title from (surrendered title through)	E8 Acquisition	E39 Actor
P28	- - - custody surrendered by (surrendered custody through)	E10 Transfer of Custody	E39 Actor
P29	- - - custody received by (received custody through)	E10 Transfer of Custody	E39 Actor
P96	- - by mother (gave birth)	E67 Birth	E21 Person
P99	- - dissolved (was dissolved by)	E68 Dissolution	E74 Group
P143	- - joined (was joined by)	E85 Joining	E39 Actor
P144	- - joined with (gained member by)	E85 Joining	E74 Group
P145	- - separated (left by)	E86 Leaving	E39 Actor
P146	- - separated from (lost member by)	E86 Leaving	E74 Group
P151	- - was formed from (participated in)	E66 Formation	E74 Group
P16	- used specific object (was used for)	E7 Activity	E70 Thing

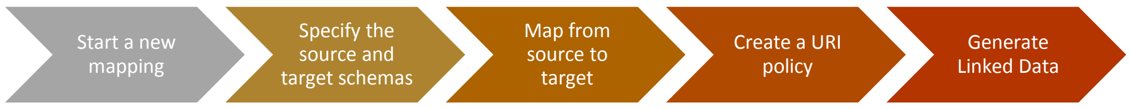
This table can be copied into an excel spreadsheet so that you can filter the table against domain, range or properties.



	A	B	C	D
1	**Property id**	**Property Name**	**Entity – Domain**	**Entity - Range**
2	P1	is identified by (identifies)	E1 CRM Entity	E41 Appellation
3	P48	- has preferred identifier (is preferred identifier of)	E1 CRM Entity	E42 Identifier
4	P78	- is identified by (identifies)	E52 Time-Span	E49 Time Appellation
5	P87	- is identified by (identifies)	E53 Place	E44 Place Appellation
6	P102	- has title (is title of)	E71 Man-Made Thing	E35 Title
7	P131	- is identified by (identifies)	E39 Actor	E82 Actor Appellation
8	P149	- is identified by (identifies)	E28 Conceptual Object	E75 Conceptual Object Appellation
9	P2	has type (is type of)	E1 CRM Entity	E55 Type
10	P137	- exemplifies (is exemplified by)	E1 CRM Entity	E55 Type
11	P3	has note	E1 CRM Entity	E62 String
12	P79	- beginning is qualified by	E52 Time-Span	E62 String
13	P80	- end is qualified by	E52 Time-Span	E62 String
14	P4	has time-span (is time-span of)	E2 Temporal Entity	E52 Time-Span
15	P5	consists of (forms part of)	E3 Condition State	E3 Condition State
16	P7	took place at (witnessed)	E4 Period	E53 Place
17	P26	- moved to (was destination of)	E9 Move	E53 Place
18	P27	- moved from (was origin of)	E9 Move	E53 Place
19	P8	took place on or within (witnessed)	E4 Period	E18 Physical Thing

7 3M Concepts

The 3M workflow that leads to Linked Data generation (RDF or Resource Description Framework) is this.


Start a new mapping
Specify the source and target schemas
Map from source to target
Create a URI policy
Generate Linked Data

It is important to note that although 3M supports this process and will continue to be developed, it represents a single tool which is part of a wider set of activities that require the interaction of people. Software alone is not sufficient to generate quality contextual data mapping.

7.1 Mapping Community

3M is a community web application. While it supports security for editing files it allows the whole community to view a mapping file. This means that you can look at other users mapping definitions to learn techniques and see approaches to different types of information. You can also share editing rights with others. Although only one person can edit at a time this provides some capability to share the mapping process with someone else online – perhaps to help, or to make corrections. The longer term objective of 3M is to provide a knowledge base of mappings (CRM patterns).

7.2 What does a mapping consist of?

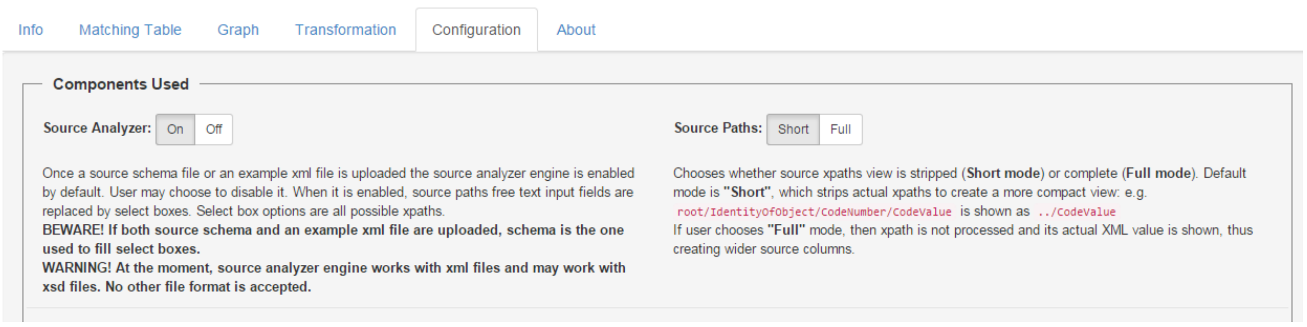
An XML source schema – This can be derived from the XML data that is exported from an information system. The export should be consistent so that if changes are made to the data the XML export remains the same. It is inevitable that XML source schemas change over time – for example to accommodate new information. When this happens, adjustments to the mapping and re-publication will be required.

A URI Generator policy File (XML) – The 3M system supports URI Generators, which are templates for creating an URI. These templates are defined in an XML file.

Target Schema(s) – The 3M system supports schema that are defined in different forms (see The Configuration Tab is used to configure various 3MEditor options. Options so far:

Source Analyser

The source analyser option is simply to choose whether you want to choose source nodes from a pull down of available elements, or whether you simply want to type them in yourself. In order to enable the pull down, you will have to upload a source schema or an example xml file first. Even if Source Analyzer is enabled, you can always ignore its suggestions, type something different and click Enter.



Components Used

Source Analyzer: [✓ On] Off

Once a source schema file or an example xml file is uploaded the source analyzer engine is enabled by default. User may choose to disable it. When it is enabled, source paths free text input fields are replaced by select boxes. Select box options are all possible xpaths.
BEWARE! If both source schema and an example xml file are uploaded, schema is the one used to fill select boxes.
WARNING! At the moment, source analyzer engine works with xml files and may work with xsd files. No other file format is accepted.

Source Paths: [✓ Short] Full

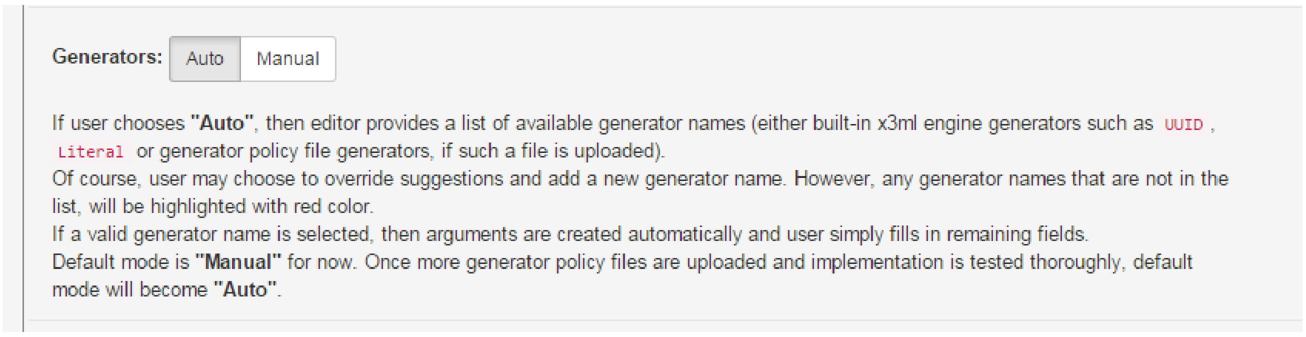
Chooses whether source xpaths view is stripped (**Short mode**) or complete (**Full mode**). Default mode is **"Short"**, which strips actual xpaths to create a more compact view: e.g. `root/IdentityOfObject/CodeNumber/CodeValue` is shown as `../CodeValue`
If user chooses **"Full"** mode, then xpath is not processed and its actual XML value is shown, thus creating wider source columns.

Source Paths

By default, source paths are stripped and a more compact view is presented to the user. However, sometimes users prefer to show the entire xpath. This option allows the user to choose between “short” and “full” paths.

Generators

The generators option is simply to choose whether you want to choose generator (instance or label) names from a pull down of available generators, or whether you simply want to type them in yourself. Available generators are either built-in or provided by an uploaded generator policy file. Even if “Generators” is set to “Auto” , you can always ignore its suggestions, type something different and click Enter.


Generators: Auto Manual

If user chooses **"Auto"**, then editor provides a list of available generator names (either built-in x3ml engine generators such as `UUID` , `Literal` or generator policy file generators, if such a file is uploaded).
Of course, user may choose to override suggestions and add a new generator name. However, any generator names that are not in the list, will be highlighted with red color.
If a valid generator name is selected, then arguments are created automatically and user simply fills in remaining fields.
Default mode is **"Manual"** for now. Once more generator policy files are uploaded and implementation is tested thoroughly, default mode will become **"Auto"**.

Target Analyser).

RDFS is the acronym for Resource Definition Framework Schema. The namespace is http://www.w3.org/2000/01/rdf-schema#. It includes, for example, **rdfs:label**. Other target schemata include:

SKOS – this defines a schema including properties such as **skos:prefLabel** (the preferred label for a concept), narrower, broader, scope note, scheme, and so on. The SKOS namespace is http://www.w3.org/2004/02/skos/core#. It includes, for example, **skos:prefLabel**.

These target schema will usually be defined with a namespace. Most schemas or ontologies are defined with a web address. These can be shortened by a name space. For example, the SKOS schema is available at http:// http://www.w3.org/2004/02/skos/core#. The namespace of SKOS is used to replace the main URI address so that http://www.w3.org/2004/02/skos/core#prefLabel becomes **skos:prefLabel**.

A Mapping Definition File (X3ML) - When the source scheme is mapped to the target schemas it produces a mapping definition file that describes the mapping. It also includes information about how to create the URI's for the nodes that will be generated. Details of X3ML are described below. Mapping from source to target schemas requires a knowledge of the source schema and the target schema.

Mapping files can be exported from 3M and they will be zipped into a compressed file. These files can be re-imported back into 3M if required creating a new mapping instance. (See export and import on the **more** menu).

7.3 X3ML

The 3M tool is a software application designed to support data mapping. It needs a mechanism for recording the alignment between source and target schemas. X3ML is an XML schema (Extensible Markup Language) designed to support the processes and functions of the Synergy data provisioning reference. It provides a clear definition of the mapping between the source schema and the target schema including the relationship and other logic to support a robust transformation of data. It records these things in a way as clear as possible so that it is even possible to understand a mapping from the source X3ML file and see exactly how 3M processes a mapping definition. 3M itself currently works by mapping from a XML source, usually an export of data from a source information system.

The following example shows the different parts of the X3ML schema:

```
<mapping>
      <domain>
        <source_node>//adlibXML/recordList/record</source_node>
        <target_node>
          <entity>
            <type>crm:E22_Man-Made_Object</type>
            <instance_generator name="PersistentId">
              <arg name="id" type="xpath">PersistentIdentifier/text()</arg>
            </instance_generator>
          </entity>
        </target_node>
      </domain>
      <link>
        <path>
          <source_relation>
            <relation>priref</relation>
          </source_relation>
          <target_relation>
            <relationship>crm:P1_is_identified_by</relationship>
          </target_relation>
        </path>
        <range>
          <source_node>priref</source_node>
          <target_node>
            <entity>
              <type>crm:E42_Identifier</type>
              <instance_generator name="PersistentIdThing">
                <arg name="persistent" type="xpath">../PersistentIdentifier/text()</arg>
                <arg name="thing" type="constant">priref</arg>
              </instance_generator>
            </entity>
          </target_node>
        </range>
      </link>
</mapping>
```

The **mapping** element denotes the start of a mapping. A domain is the CRM entity from which relationships to other CRM entities originate. In this example the domain is the object itself. The object record (in this case identified with the source element 'record') has a source node of;

adlibXML/recordList/record

This is the schema element in the source XML which represents the object record and which is contained in the X3ML <**source_node>** element. This is followed by the **<target_node>** – the element in the CRM that denotes the object. In this case **E22_Man-made_Object**. In this example the target has also been assigned a **URI** which is created using the <**instance_generator>** node. Each instance generator has a name and some arguments – more later.

Once the domain is specified links are then attached to the domain which link to other entities. The link or relationship has a source, in this case **priref**. **priref** is an XML element **<priref>** in the source file which is immediately below the domain path, e.g., **adlib/recordlist/record/priref. Priref** is relative to the domain path, **adlib/recordlist/record**.

The target relationship is **P1_is_identified_by** – so priref in the source is mapped firstly to a relationship **P1_is_identified_by**.

Priref is also mapped to the target node. It is not just the field or element that determines the relationship but is also the field that determines the target node. The target node is **E42_Identifier**.

Again this node is, in another part of 3M, assigned a URI through an **instance generator**. In this case part of the instance generator uses the value inside the priref element. The statement **../PersistentIdentifier/text()** means, go back up the hierarchical one place (i.e. from **adlib/recordlist/record/priref** to **adlib/recordlist/record** and then go to **adlib/recordlist/record/PersistentIdentifier** and get the text, or the value, inside that element.

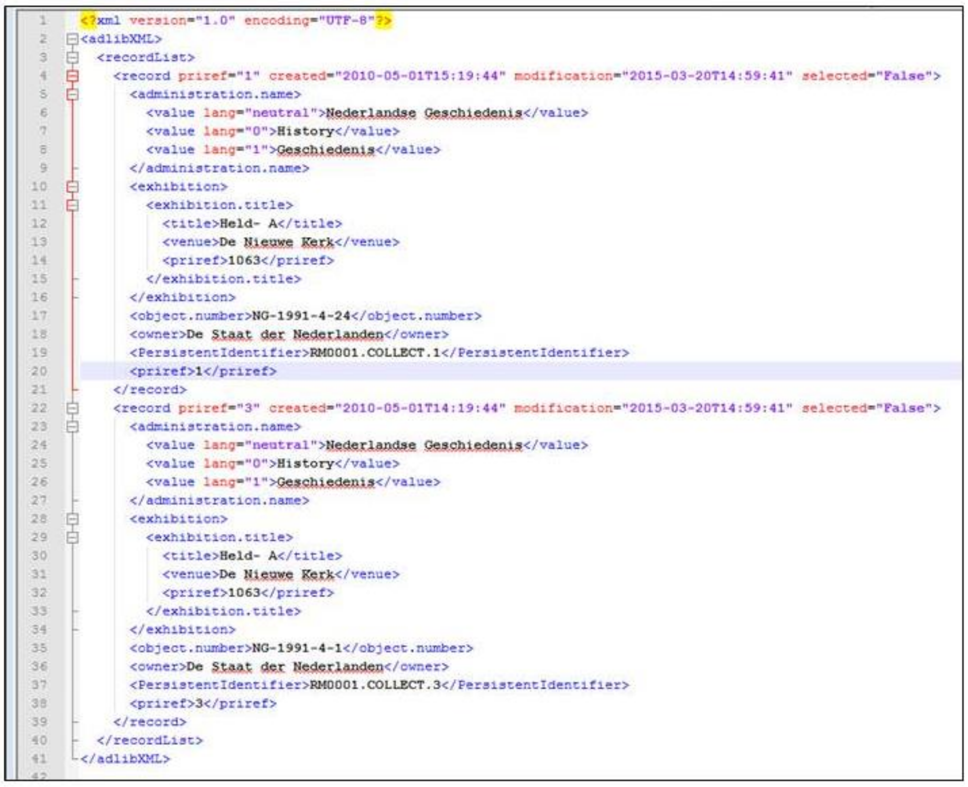
These element addresses are explained in more detail below – see XPATH. Any number of links may be associated with a particular domain.

7.4 XPATH

3M is a developing system ultimately designed to be used by non-technical users. However, in its current iteration not all the technical elements of mapping have been eliminated. The mapping system and URI generator rely on a basic knowledge of XPath. XPath is a W3C (World Wide Web Consortium) recommendation that uses path expressions to navigate through XML documents. It actually provides a syntax for defining parts of an XML document. XML documents are treated as trees of nodes. The topmost element of the tree is called the root element. A detailed tutorial on XPath is available in http://www.w3schools.com/xsl/xpath_intro.asp

XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions, listed in http://www.w3schools.com/xsl/xpath_syntax.asp:

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes



```xml
<?xml version="1.0" encoding="UTF-8"?>
<adlibXML>
  <recordList>
    <record priref="1" created="2010-05-01T15:19:44" modification="2015-03-20T14:59:41" selected="False">
      <administration.name>
        <value lang="neutral">Nederlandse Geschiedenis</value>
        <value lang="0">History</value>
        <value lang="1">Geschiedenis</value>
      </administration.name>
      <exhibition>
        <exhibition.title>
          <title>Held- A</title>
          <venue>De Nieuwe Kerk</venue>
          <priref>1063</priref>
        </exhibition.title>
      </exhibition>
      <object.number>NG-1991-4-24</object.number>
      <owner>De Staat der Nederlanden</owner>
      <PersistentIdentifier>RM0001.COLLECT.1</PersistentIdentifier>
      <priref>1</priref>
    </record>
    <record priref="3" created="2010-05-01T14:19:44" modification="2015-03-20T14:59:41" selected="False">
      <administration.name>
        <value lang="neutral">Nederlandse Geschiedenis</value>
        <value lang="0">History</value>
        <value lang="1">Geschiedenis</value>
      </administration.name>
      <exhibition>
        <exhibition.title>
          <title>Held- A</title>
          <venue>De Nieuwe Kerk</venue>
          <priref>1063</priref>
        </exhibition.title>
      </exhibition>
      <object.number>NG-1991-4-1</object.number>
      <owner>De Staat der Nederlanden</owner>
      <PersistentIdentifier>RM0001.COLLECT.3</PersistentIdentifier>
      <priref>3</priref>
    </record>
  </recordList>
</adlibXML>
```

Figure 5: source XML excerpt

In Figure 5 an excerpt of a source XML document representing two records is displayed. The table below lists some desired actions in 3M, the path expressions that can be used and the result of the expressions. It should be noted that if the path starts with a slash (/) it always represents an absolute path to an element:

Desired Action in 3M	Path Expression	Result according to example of Figure X
Specify that *record* is the source Domain (nodes named *record*)	//record	Selects all nodes with the name *record* no matter where they are in the document. <record priref="1" ….> <record priref="3" ….>
Specify that *record* is the source Domain (nodes named *record*)	adlibXML/recordList/record	Selects all nodes with the name *record* that are children of adlibXML/recordList As above, the two nodes will be selected Note: this approach uses absolute paths and ensures that nested nodes with the same name will be distinguished.
Specify that *priref* is the source path (relative to the Domain)	priref	Selects all nodes *priref* that are children of *record* *<priref>1</priref>* *<priref>3</priref>*
Specify that *priref* is the source path only if it is a child of *exhibition/ exhibition.title* (relative to the Domain)	*exhibition/ exhibition.title/priref*	Selects all nodes *exhibition/ exhibition.title/priref* that are children of *record* *<priref>1063</priref>* *<priref>1063</priref>*
Specify that *priref* is the source path (relative to the Domain) nomatter where it is under *record*.	*//priref*	Selects all nodes *priref* that are children of *record* *<priref>1</priref>* *<priref>1063</priref>* *<priref>3</priref>* *<priref>1063</priref>*

Get the value of attribute *priref* of node *record*	*@priref*	Selects the attribute *priref* of the current node (*record*)
		priref="1"
		priref="3"

Specify that *value* is the source path only if it is child of *administration.name* (relative to the Domain)	administration.name/value	Selects all value elements that are children of administration.name
Specify an if condition based on the value of the attribute *lang* in the path *administration.name/value*	administration.name/value/@lang	Selects the attribute *lang* in node *value* child of *administration.name* (relative to *record)* *lang="neutral"* *lang="0"* *lang="1"*

		lang="neutral" *lang="0"* *lang="1"*
Get the value of node *value* if its attribute *lang* has the value "*neutral*"	value[@lang="neutral"]/text()	Text = Nederlandse Geschiedenis Text = Nederlandse Geschiedenis

7.4.1 Where is XPath used?

Source Nodes

In the mapping table source nodes can be inserted relative to the domain node. If the domain path is **adlibXML/recordList/record** and the source node for a link is **adlibXML/recordList/record/priref**, then the source node could simply say **priref.** Occasionally the source node might not be a direct descendent of the domain and some XPath may be required.

Rules

The testing of data using rules often needs XPath statements since a mapping may dependent on a value attached to a different element than the source node.

Instance Generator

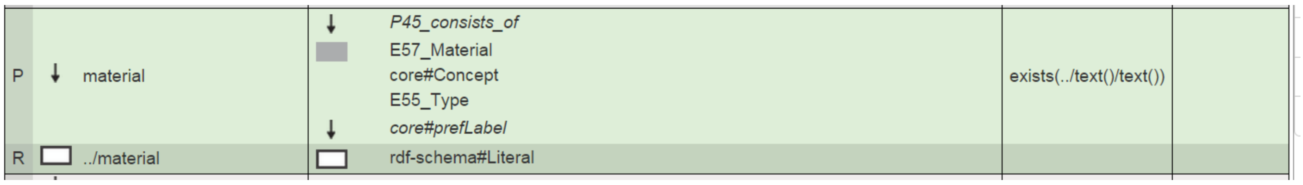
XPath statements are also used in generating URIs where the data in the source is used to form Uniform Resource Identifiers.

7.5 Simple Knowledge Organisation System (SKOS)

SKOS is a specification and schema supporting concept terms such as those used in thesauri, classification schemes, subject heading systems and taxonomies within the framework of the Semantic Web. The aim is not to replace original conceptual vocabularies but allow them to be ported as Linked Data enabling wide re-use and better interoperability using the web.

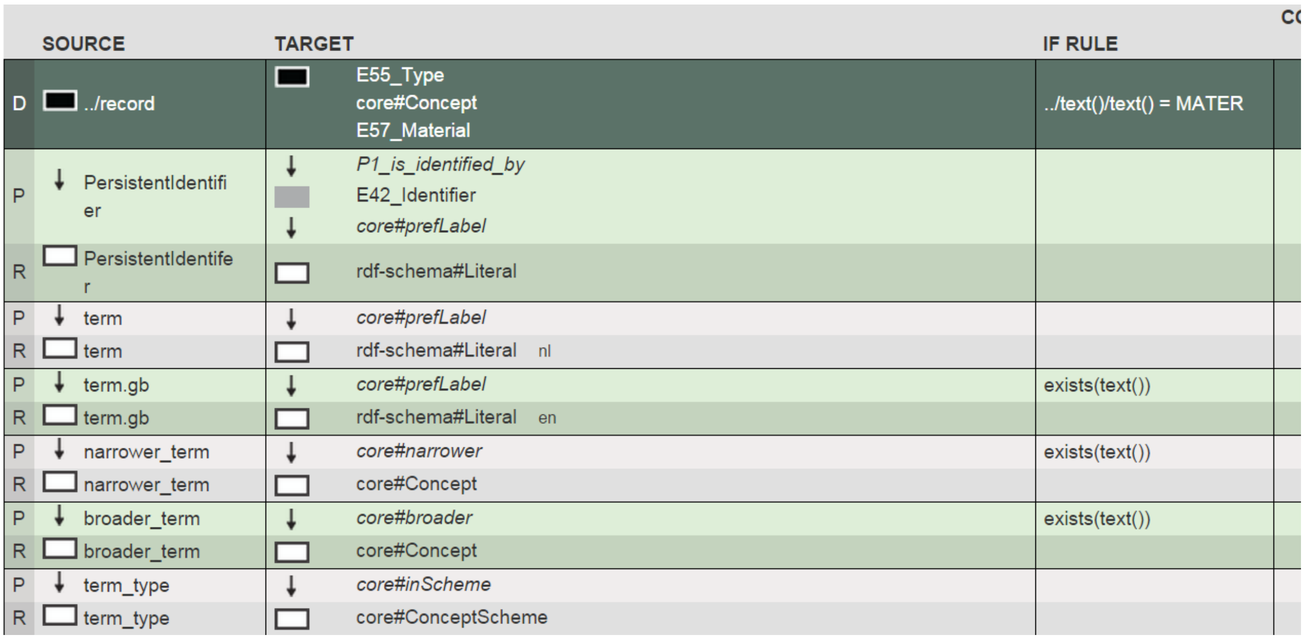
Typically you would start a new mapping file for concepts. This means that in the core file you may end a mapping with E55_Type and assign a URI, for example, http://collection.rijksmuseum.org/id/thesauri/12245. Another mapping file using the same URI scheme picks up from the main mapping file.

Here is the main mapping on the domain **E22_Man-Made_Object.** The **concept** has a **prefLabel** (a literal label) **but** this isn't really necessary and will be duplicated alongside other SKOS properties in a separate mapping file.



P	↓	material	↓ ▨ ↓	*P45_consists_of* E57_Material core#Concept E55_Type *core#prefLabel*	exists(../text()/text())	
R	☐	../material	☐	rdf-schema#Literal		

The separate mapping file for Materials looks like this;



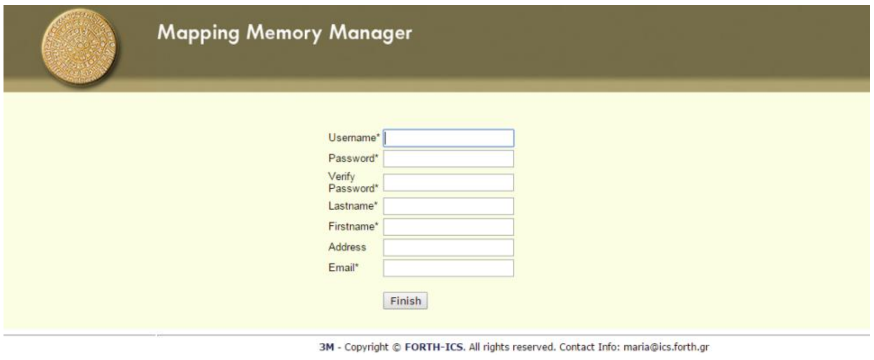
	SOURCE		TARGET		IF RULE	CC
D	■ ../record	■	E55_Type core#Concept E57_Material		../text()/text() = MATER	
P	↓ PersistentIdentifi er	↓ ▨ ↓	*P1_is_identified_by* E42_Identifier *core#prefLabel*			
R	☐ PersistentIdentife r	☐	rdf-schema#Literal			
P	↓ term	↓	*core#prefLabel*			
R	☐ term	☐	rdf-schema#Literal	nl		
P	↓ term.gb	↓	*core#prefLabel*		exists(text())	
R	☐ term.gb	☐	rdf-schema#Literal	en		
P	↓ narrower_term	↓	*core#narrower*		exists(text())	
R	☐ narrower_term	☐	core#Concept			
P	↓ broader_term	↓	*core#broader*		exists(text())	
R	☐ broader_term	☐	core#Concept			
P	↓ term_type	↓	*core#inScheme*			
R	☐ term_type	☐	core#ConceptScheme			

The domain is the range of the previous mapping in **E22_Man-Made_Object** and the links just connect to that domain. Clearly this could have been defined in the main mapping file, but separating them reduces the size of the main mapping file and makes things clearer.

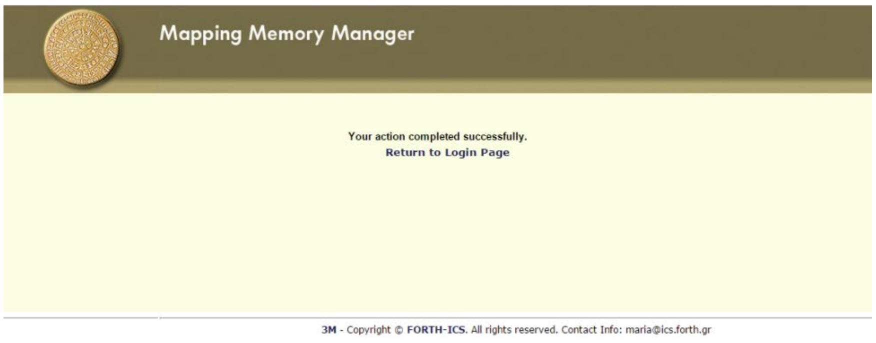
An example is the term "gold". This has an identifier representing the concept, with a **prefLabel** that is defined in both Dutch and English ('goud' and 'gold'). We used the "Add instance info" option to define the languages for each as 'en' and 'nl'. A key component of SKOS is the ability to represent hierarchies using broader and narrower terms, enabling the hierarchical links. This provides relationships to other concepts, for example, a concept with the label, '14 carat gold' (narrower) and a broader term could be a concept with the label, 'transition metal'. Terms belong to a concept scheme (a particular authority) and different concept schemes from different organisations may appear in an aggregated dataset but can also be linked to other schemes, doe example, using the skos property, 'exactMatch'. Representation in SKOS can also be associative (non-hierarchical) links, such as the relationship between one type of event and a category of entities which typically participate in it.

8 Using the FORTH 3M Service

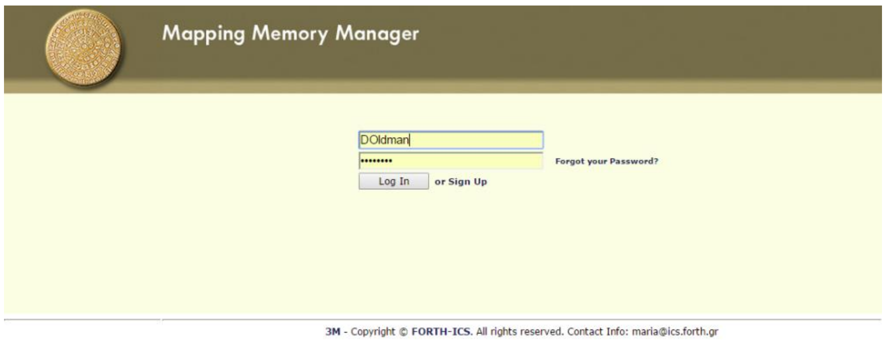
The 3M implementation at FORTH can be accessed using a normal Web Browser at http://www.ics.forth.gr/isl/3M/, and is free to use. However, you must first register an account to get a username and password. From the login screen choose the sign up link and complete the form below. Fields with an '*' are mandatory.



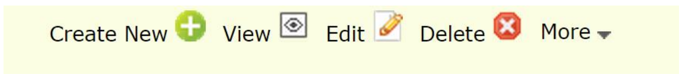
If successful you will be invited back to the login screen


Mapping Memory Manager
Your action completed successfully.
Return to Login Page
3M - Copyright © FORTH-ICS. All rights reserved. Contact Info: maria@ics.forth.gr

You can then logon with your username and password immediately.


Mapping Memory Manager
DOldman
••••••••
Forgot your Password?
Log In
or Sign Up
3M - Copyright © FORTH-ICS. All rights reserved. Contact Info: maria@ics.forth.gr

9 Creating a new Mapping



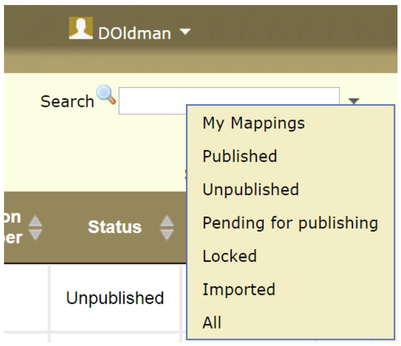
9.1 Create

When you create a new mapping you are asked to give it a title, select one or more of the proposed Target Schema and then click **Finish**. At least one schema must be selected (normally CIDOC CRM). 3M proposes a list of Target Schemata but it is not restricted to these. Later on, the user can load more Target Schemata (see Info Tab).

The proposed Target Schemata include CIDOC CRM and a suite of CRM extensions suitable for specific applications (http://www.ics.forth.gr/isl/index_main.php?l=e&c=229).

- CIDOC CRM v6.0 – the core ontology
- CRMdig 3.2 – a model for provenance metadata
- CRMgeo 1.2 – a spatiotemporal model
- CRMsci 1.2. 2 – a scientific observation model
- CRMarchaeo 1.2.1 – an excavation model
- FRBR 2.1 – modelling the new library practice of IFLA
- CRMext4SKOSandLabel 1.2 – some of the basic properties of SKOS (Simple Knowledge Organisation System) used to provide authority information for E55 Types (see later), for example **prefLabel, broader, narrower**, **scopenote**; and the RDFS schema which provides useful properties like **rdfs:label**, used for simple string values.
- CRMpc 1.0 – an implementation of the *properties of properties* (the ".1" properties) of CIDOC CRM in rdf

If a new mapping is created successfully then the id of the new mapping is displayed. You can use the mappings link to return to the main screen.



9.2 Finding your mapping

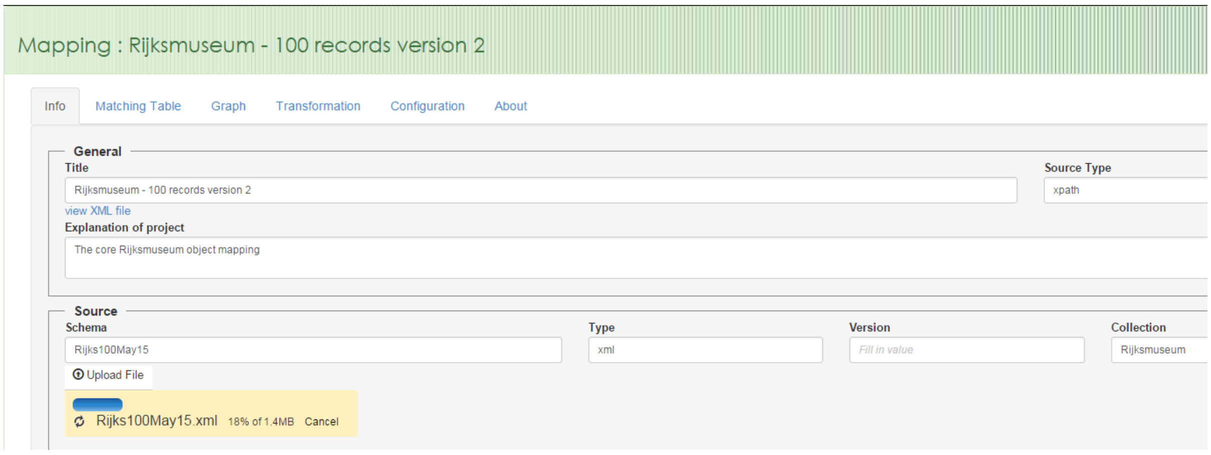
The number of mappings on the FORTH implementation of 3M is increasing. However, you can search for a mapping (title, id, username) and you can also sort and filter the mappings. This includes the ability to filter down to the mappings that you are able to edit – "your mappings". Use the pulldown symbol next to the search function to do this.

9.3 Configuring your Mapping

Select your mapping from the main screen and choose **Edit** from the main menu. The mapping definition area has a number of tabs. These are:

9.3.1 Info Tab

This includes project information about the mapping and the people involved in it, the source schema, the target schemas, the sample data file, the generator policy file. Clicking Edit on the Info screen allows you to add information about your mapping and add source and target schemas.


Mapping : Rijksmuseum - 100 records version 2

General

Title: Rijksmuseum - 100 records version 2

Source Type: xpath

view XML file

Explanation of project: The core Rijksmuseum object mapping

Source

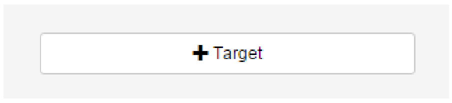
Schema	Type	Version	Collection
Rijks100May15	xml	Fill in value	Rijksmuseum

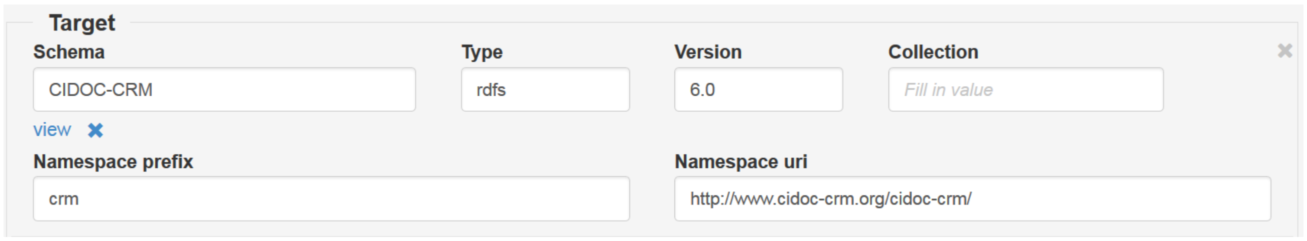
Upload File

Rijks100May15.xml 18% of 1.4MB Cancel

Give you mapping a different title if you need to and provide some additional description about your mapping project. You can upload a source schema, this might be an example XML file or an XML schema file. Valid formats are, XML (a data file), XSD (a XML schema file). This allows you to reference the element names from the source XML in the Matching Table when picking Source Nodes. (Note that source analyser must be set to On. This setting is found on the configuration tab)

You can also add target schema, either different ones or news versions of the ones selected during the mapping creation.




Target

Schema	Type	Version	Collection
CIDOC-CRM	rdfs	6.0	Fill in value

view

Namespace prefix: crm

Namespace uri: http://www.cidoc-crm.org/cidoc-crm/

When you add a target schema you should fill in the information – the schema name, the type of file, the version of the schema, the namespace prefix and the full namespace uri.

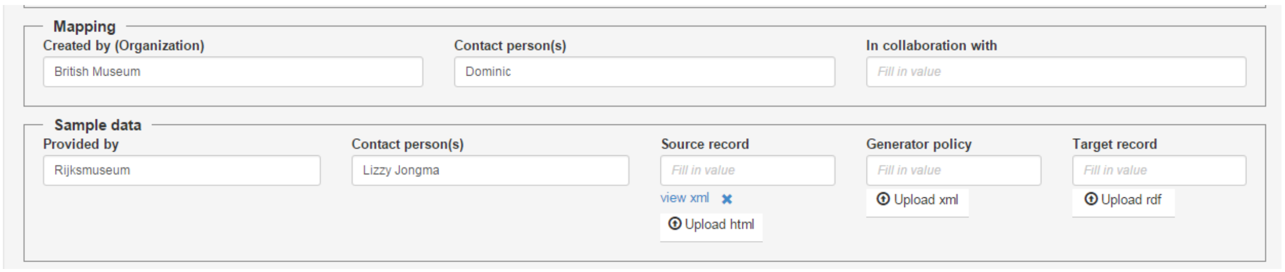
The prefix is a short name that replaces the full URI name. In the example instead of referring to an ontology class or property like this –

http://www.cidoc-crm.org/cidoc-crm/E22_Man-Made_Object

…it can be represented as

crm:E22_Man-Made_Object.

Finally, some additional information can be provided including sample data, for example, the same source data (XML) or an HTML example, a generator policy XML file (used to create URIs) and when the mapping is complete and processed, the resulting Linked data file.


Mapping

Created by (Organization)	Contact person(s)	In collaboration with
British Museum	Dominic	Fill in value

Sample data

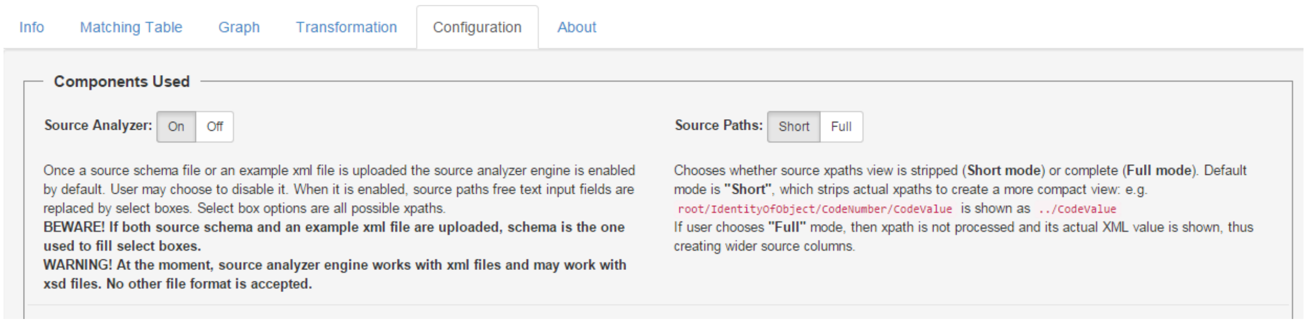
Provided by	Contact person(s)	Source record	Generator policy	Target record
Rijksmuseum	Lizzy Jongma	Fill in value	Fill in value	Fill in value
		view xml / Upload html	Upload xml	Upload rdf

9.3.2 Configuration Tab

The Configuration Tab is used to configure various 3MEditor options. Options so far:

Source Analyser

The source analyser option is simply to choose whether you want to choose source nodes from a pull down of available elements, or whether you simply want to type them in yourself. In order to enable the pull down, you will have to upload a source schema or an example xml file first. Even if Source Analyzer is enabled, you can always ignore its suggestions, type something different and click Enter.



Source Analyzer: [✓ On] Off

Once a source schema file or an example xml file is uploaded the source analyzer engine is enabled by default. User may choose to disable it. When it is enabled, source paths free text input fields are replaced by select boxes. Select box options are all possible xpaths.
BEWARE! If both source schema and an example xml file are uploaded, schema is the one used to fill select boxes.
WARNING! At the moment, source analyzer engine works with xml files and may work with xsd files. No other file format is accepted.

Source Paths: [✓ Short] Full

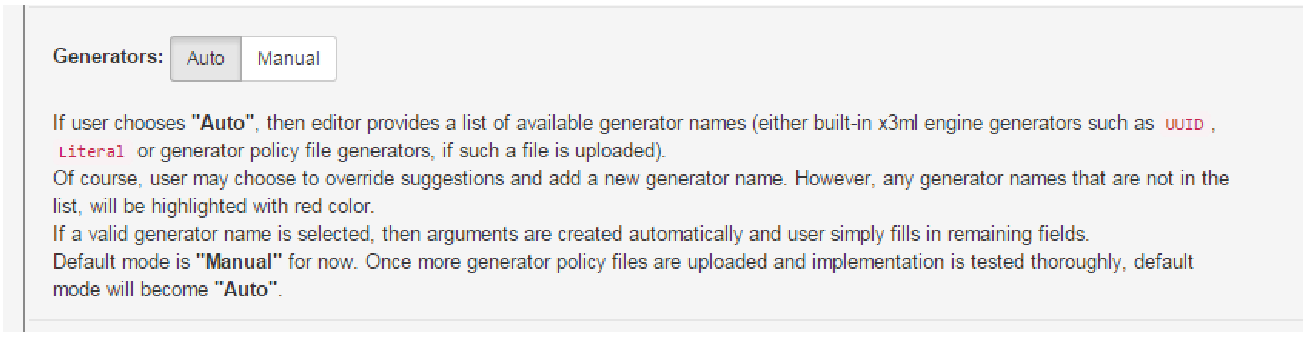
Chooses whether source xpaths view is stripped (**Short mode**) or complete (**Full mode**). Default mode is **"Short"**, which strips actual xpaths to create a more compact view: e.g. `root/IdentityOfObject/CodeNumber/CodeValue` is shown as `../CodeValue`
If user chooses **"Full"** mode, then xpath is not processed and its actual XML value is shown, thus creating wider source columns.

Source Paths

By default, source paths are stripped and a more compact view is presented to the user. However, sometimes users prefer to show the entire xpath. This option allows the user to choose between “short” and “full” paths.

Generators

The generators option is simply to choose whether you want to choose generator (instance or label) names from a pull down of available generators, or whether you simply want to type them in yourself. Available generators are either built-in or provided by an uploaded generator policy file. Even if “Generators” is set to “Auto” , you can always ignore its suggestions, type something different and click Enter.



Generators: [✓ Auto] Manual

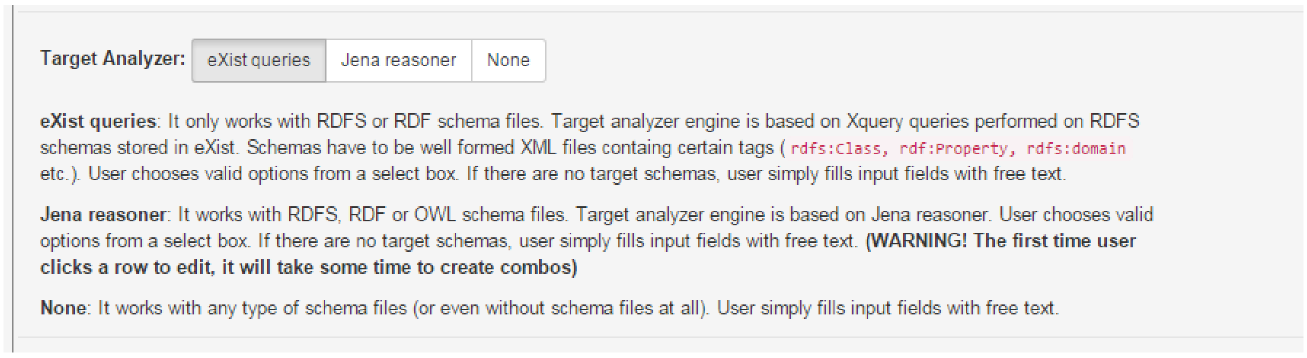
If user chooses **"Auto"**, then editor provides a list of available generator names (either built-in x3ml engine generators such as `UUID` , `Literal` or generator policy file generators, if such a file is uploaded).
Of course, user may choose to override suggestions and add a new generator name. However, any generator names that are not in the list, will be highlighted with red color.
If a valid generator name is selected, then arguments are created automatically and user simply fills in remaining fields.
Default mode is **"Manual"** for now. Once more generator policy files are uploaded and implementation is tested thoroughly, default mode will become **"Auto"**.

Target Analyser

Generally, there is no restriction on the target schema. However, in the current implementation, the Target Analyser understand only CIDOC CRM and compatible schemata. Other target schemata can be used without the help of the Target Analyser.

3M uses the RDF encoding of CIDOC CRM version 6.0. Note that this is NOT a definition of the CIDOC CRM, but an encoding derived from the authoritative release of the CIDOC CRM v6.0 on http://www.cidoc-crm.org/official_release_cidoc.html

The target analysers use different methods for representing schemas (ontologies) when the user clicks on a target pull down list of properties and entities. The eXist analyser is the default choice. It works with RDFS and RDF files, which contain specific tags. Increasingly, ontologies are being produced in other forms that are not XML (turtle, ntriples). Therefore another reasoner was added (the Jena reasoner) so that other RDF formats can be used. These tools allow the Editor to display only the properties and entities that are valid with a particular mapping context.



eXist queries: It only works with RDFS or RDF schema files. Target analyzer engine is based on Xquery queries performed on RDFS schemas stored in eXist. Schemas have to be well formed XML files containg certain tags (`rdfs:Class, rdf:Property, rdfs:domain` etc.). User chooses valid options from a select box. If there are no target schemas, user simply fills input fields with free text.

Jena reasoner: It works with RDFS, RDF or OWL schema files. Target analyzer engine is based on Jena reasoner. User chooses valid options from a select box. If there are no target schemas, user simply fills input fields with free text. **(WARNING! The first time user clicks a row to edit, it will take some time to create combos)**

None: It works with any type of schema files (or even without schema files at all). User simply fills input fields with free text.

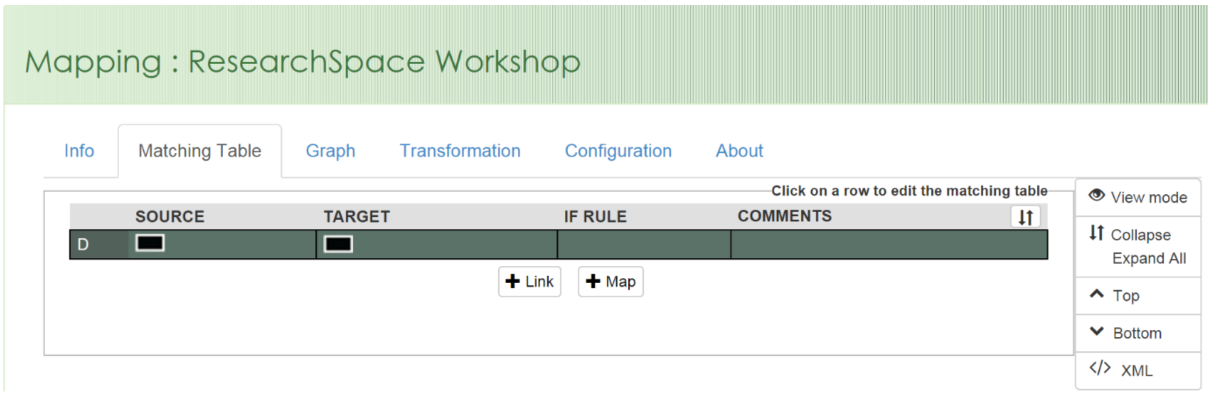
If you wish you can turn these off and simply type in what you want manually.

"Mapping Suggester" – Roadmap

It is the intention that 3M provides a system of mapping suggestions for the CIDOC CRM ontology. This is work that has started but will take time to evolve into a simple user interface.

9.3.3 Matching Table

When you first look at a new matching table it always starts with an empty template which requires the first **Domain** 'D'.



The columns are:

Source - The specific source element from the source XML.

Target - The specific part of the target schema to match with the source.

If Rule - The ability to add some simple logic to the mapping.

Comments - The ability to create some notes about an individual mapping.

A small control panel provides some ancillary functionality.

View mode collapses any editing views to get a clearer view of the mapping.

Collapse and Expand All toggles between a just seeing the mapping domains and links that have been defined under them

Top takes you to the top (domain) of the matching table.

Bottom takes you to the bottom (last link) of the matching table.

XML provides a popup window showing the X3ML that is being produced. When in edit mode, it only shows the X3ML for the part that is edited. Otherwise, user is provided with X3ML for the entire tab in view.

9.3.3.1 Domain, Property and Range

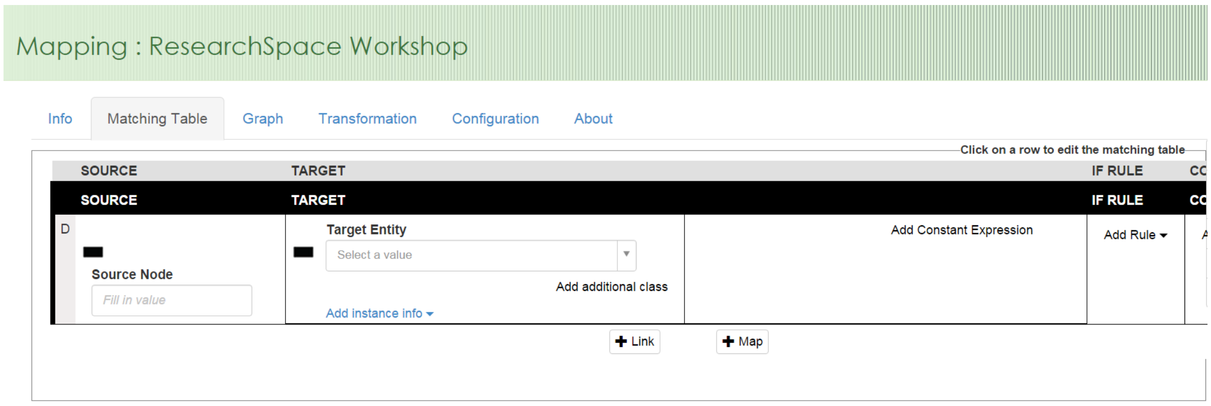
The other elements of a mapping are the **Property** (or **Relationship**) and the **Range**. When relationships are attached (Linked) to a **Domain** the **Property** and **Range** are added to complete the relationship from a **Domain** to another **Entity.**

For example, the target Domain, **Man-Made Object,** may use a source property (field/element name) "object_id" as the basis for a target property, **is identified by,** and a target range which is the entity **Identifier**.

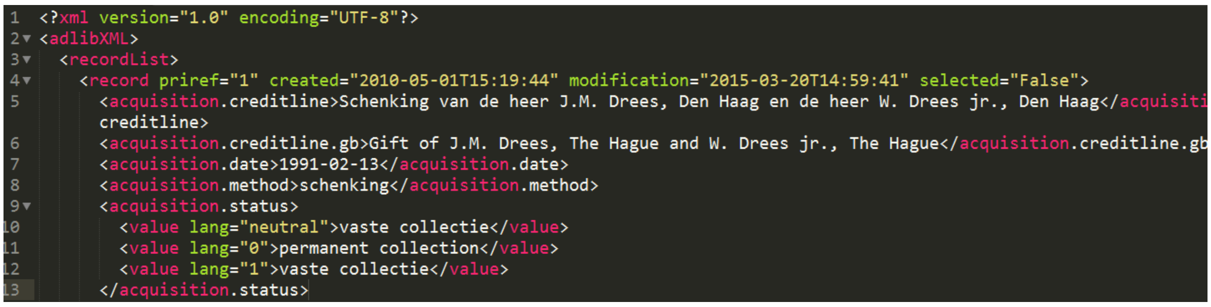
Once you have started your mapping and have an initial **Domain**, other **Domains** may be originated by being target entities creating a hierarchy of domains which can be many levels – reflecting the structure of the CRM . For example, the mapping from **Man-Made_Object** to **Identifier,** which may then become a domain itself for **E55_Type**.

10 Editing your Mapping

10.1 The First Domain



When you first click on the form it opens up the editor to enter the source and target information for the **Domain**. The first mapping for the first **Domain** asks for the **Source Node** and the **Target Entity**. The first Domain is typically the main Domain or focus of interest from which other properties, entities (including events) are connected. It is typically the one that overall defines the rest of the record that you are mapping. This would mean that the **Source Node** for the first domain is typically the XML element that provides the record boundaries or where an individual record starts and stops. For example, in the Rijksmuseum records, the XML starts like this.



```
<?xml version="1.0" encoding="UTF-8"?>
<adlibXML>
  <recordList>
    <record priref="1" created="2010-05-01T15:19:44" modification="2015-03-20T14:59:41" selected="False">
      <acquisition.creditline>Schenking van de heer J.M. Drees, Den Haag en de heer W. Drees jr., Den Haag</acquisiti
      creditline>
      <acquisition.creditline.gb>Gift of J.M. Drees, The Hague and W. Drees jr., The Hague</acquisition.creditline.gb
      <acquisition.date>1991-02-13</acquisition.date>
      <acquisition.method>schenking</acquisition.method>
      <acquisition.status>
        <value lang="neutral">vaste collectie</value>
        <value lang="0">permanent collection</value>
        <value lang="1">vaste collectie</value>
      </acquisition.status>
```

<adlibXML> is the root of the XML tree

<recordList> is an element that groups together a series of records

<record> is the element which contains all the elements of a single record.

Therefore **record** is the source element for the thing which will represent a primary, and most likely the first domain of the mapping. At the British Museum, for example, there is a mixture of different things (object types). These are all described as **Man-Made Objects.** However, some items in a museum might not all be "man made" and might be, for example, physical things from nature.

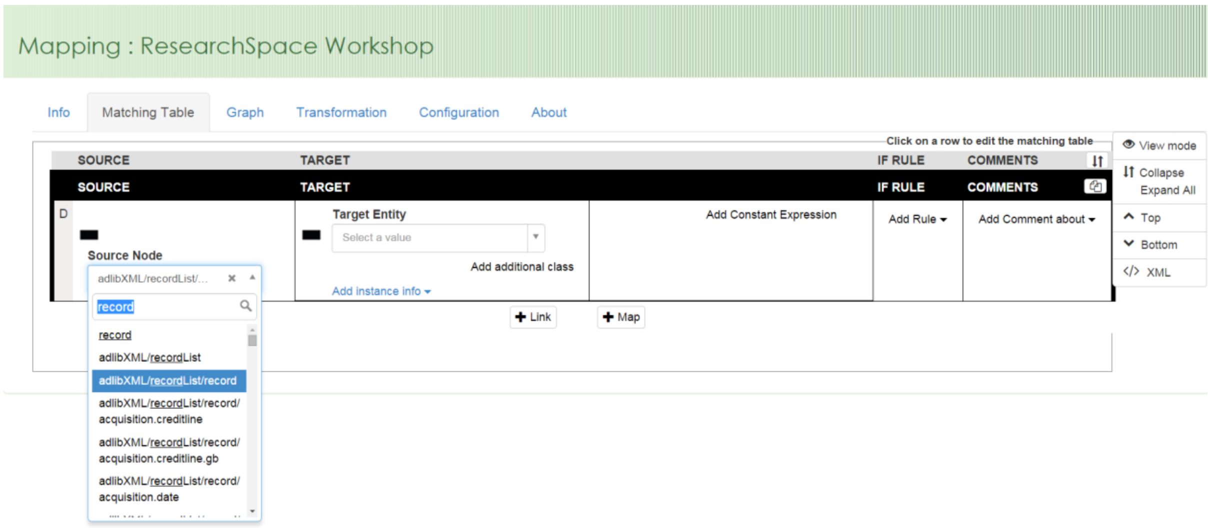
<adlibXML>

<recordList>

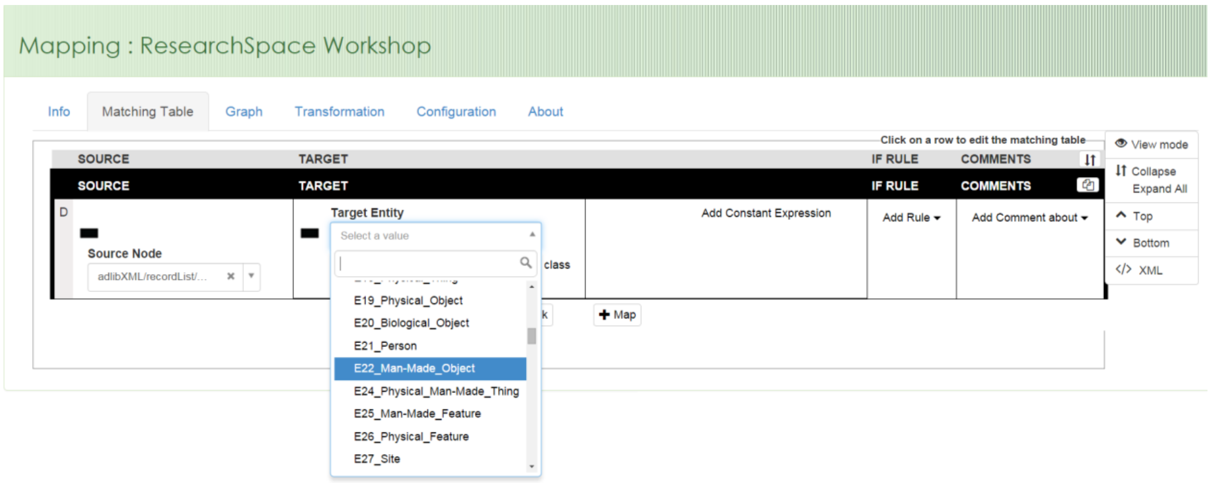
<record priref="1" created="2010-05-01T15:19:44" modification="2015-03-20T14:59:41" selected="False">

In this case I can choose the correct element from the pull down list within "**Source Node"** and choose the "record" element.

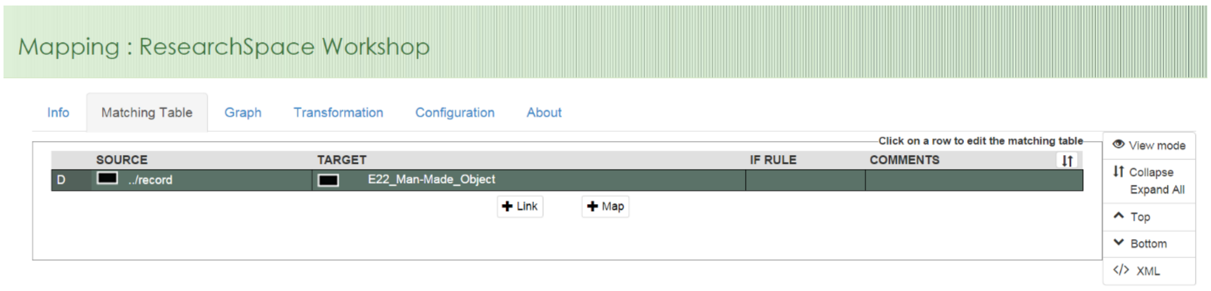
In this diagram the path **adlibXML/recordList/record** is selected.



The correct class for this domain can then be selected by choosing from the drop down list in Target Entity. In this case, **E22_Man-Made_Object**.



In **view mode** the editor is cleared and I can see the selections from Source and Target clearly.



	SOURCE	TARGET	IF RULE	COMMENTS
D	../record	E22_Man-Made_Object		

10.2 Links & Semantics

If we wanted to map a source field containing an identifier in a record, perhaps an object identifier, it would be a link from the **Domain** of **E22_Man-Made_Object**. Let's say that the field is called, '**object_identifer'.** In the British Museum different identifiers exist and sometimes a single object might have multiple identifiers.

We firstly think about the semantics both the implicit and explicit. Typically the semantics will extend beyond that which is explicit in the database. Consider the following things that we may take for granted within the organisation but which provide better context for external users.

1. The field is an identifier for the object.
2. It has a value.
3. The identifier has a type that distinguishes it from other identifiers.
4. The reason and description for the identifier.

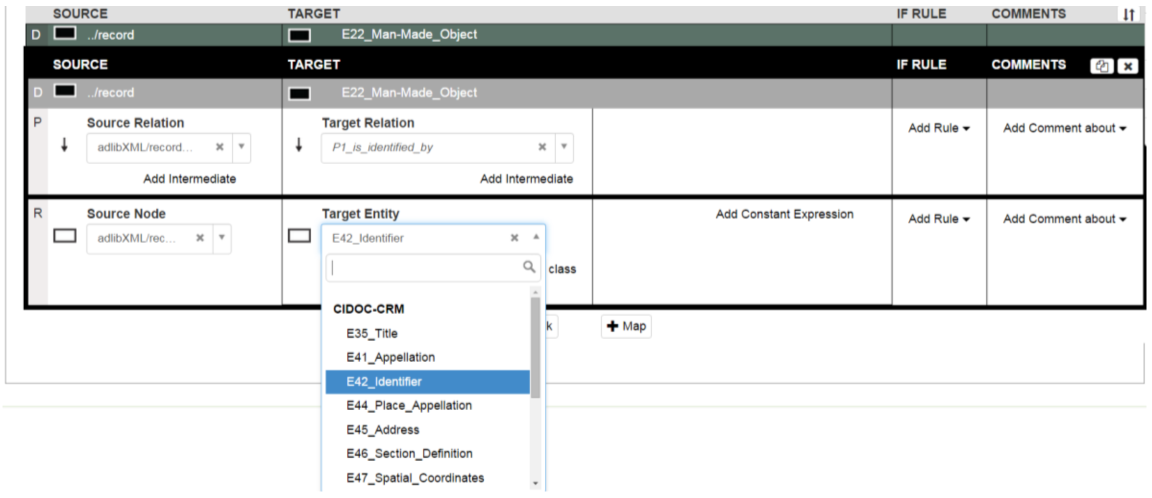
These semantic require a number of entities.

- The object entity
- The identifier entity
- The type entities
- String entities for the values

What are the relationships between these entities?

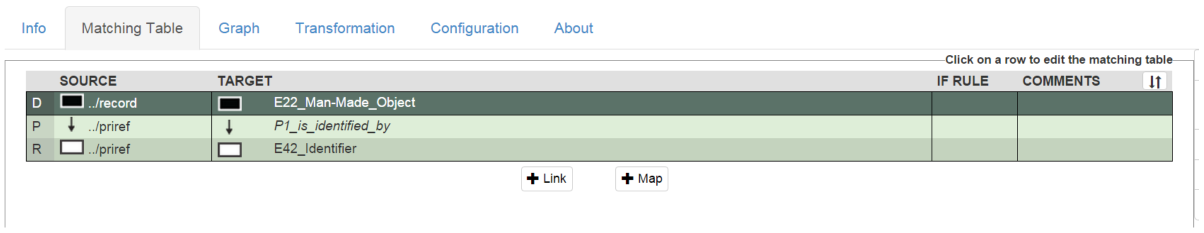
- An object is identified by an Identifier
- An identifier has a type
- A type has a value and a description
- An identifier has a value.

How are these then represented in 3M? In the following screenshot we can see the relationship between **Man-Made_Object** and **Identifier** in the CIDOC CRM ontology. Both source relation and source node are both **adlibXML/recordList/record/priref**



	SOURCE	TARGET		IF RULE	COMMENTS
D	../record	E22_Man-Made_Object			

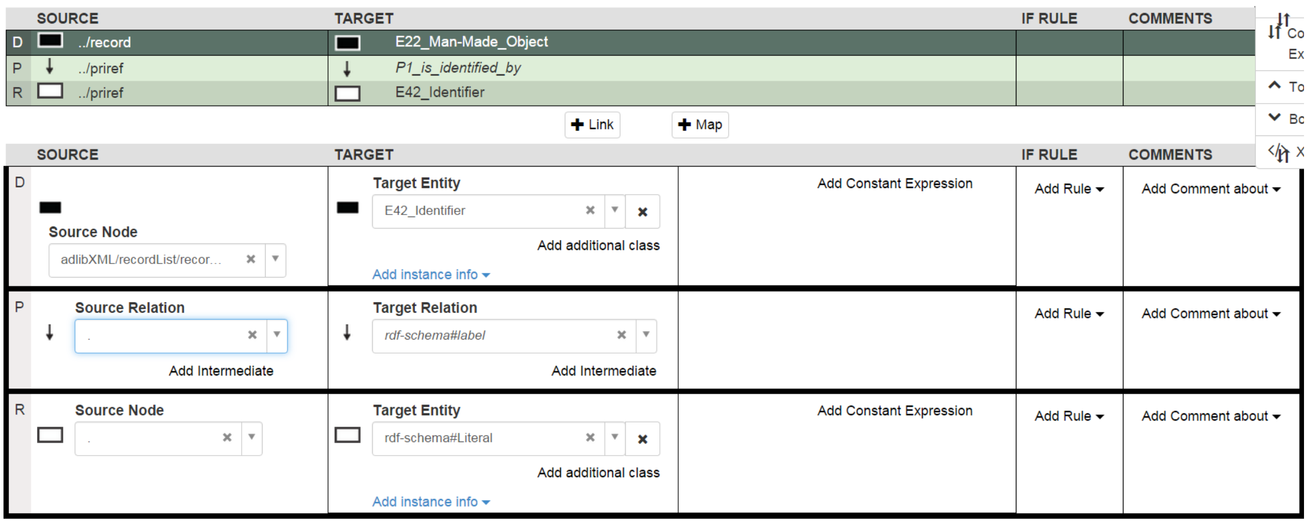
	SOURCE	TARGET		IF RULE	COMMENTS
D	../record	E22_Man-Made_Object			
P	Source Relation: adlibXML/record... Add Intermediate	Target Relation: P1_is_identified_by Add Intermediate		Add Rule	Add Comment about
R	Source Node: adlibXML/rec...	Target Entity: E42_Identifier	Add Constant Expression	Add Rule	Add Comment about



	SOURCE	TARGET	IF RULE	COMMENTS
D	../record	E22_Man-Made_Object		
P	../priref	*P1_is_identified_by*		
R	../priref	E42_Identifier		

10.3 Links & Intermediates

However, we also wanted to represent the value of the **Identifier** which has exactly the same source element. There are two ways to achieve this. The first is to establish another domain, but using **Identifier.** To do this use the + Map button and use the same path that was used for Identifier in the previous domain when it was a target in the '**Man-Made_Object'** domain.



	SOURCE	TARGET	IF RULE	COMMENTS
D	../record	E22_Man-Made_Object		
P	../priref	*P1_is_identified_by*		
R	../priref	E42_Identifier		

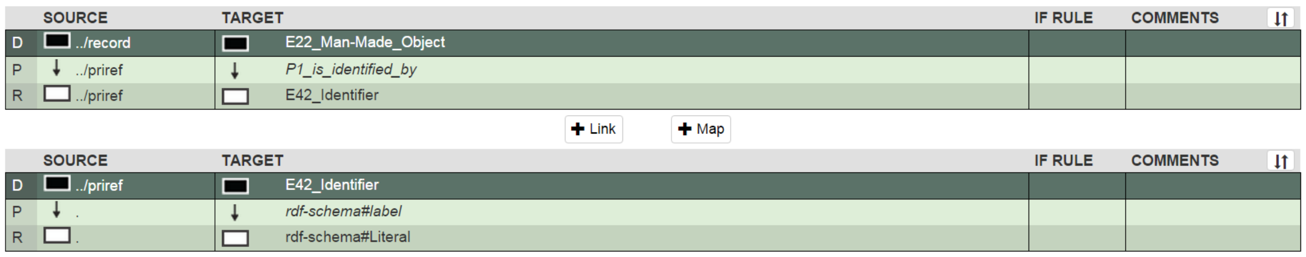
+ Link + Map

	SOURCE	TARGET		IF RULE	COMMENTS
D	Source Node: adlibXML/recordList/recor...	Target Entity: E42_Identifier; Add additional class; Add instance info	Add Constant Expression	Add Rule	Add Comment about
P	Source Relation: . ; Add Intermediate	Target Relation: *rdf-schema#label*; Add Intermediate		Add Rule	Add Comment about
R	Source Node: .	Target Entity: rdf-schema#Literal; Add additional class; Add instance info	Add Constant Expression	Add Rule	Add Comment about

In this method you create one domain for **Man-Made Object** and one domain for **Identifier**

Note: Because the creation of a Domain simply to create a label is inefficient the next version of 3M will allow labels to be inserted on the interface against an entity without the need to create another Domain.

s



	SOURCE	TARGET	IF RULE	COMMENTS
D	../record	E22_Man-Made_Object		
P	../priref	*P1_is_identified_by*		
R	../priref	E42_Identifier		

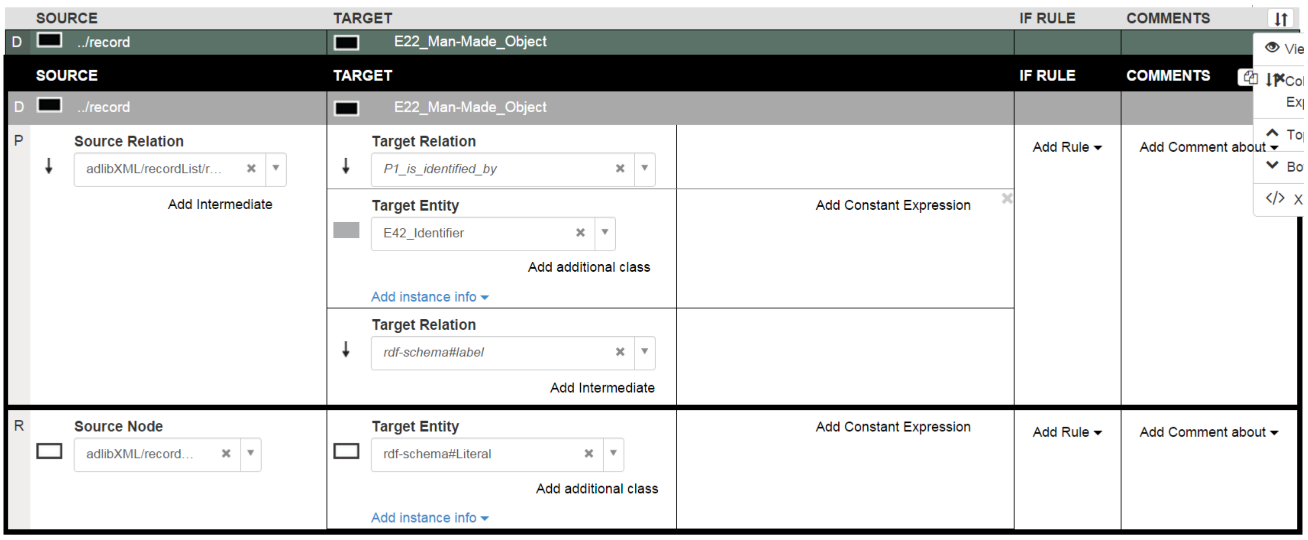
+ Link + Map

	SOURCE	TARGET	IF RULE	COMMENTS
D	../priref	E42_Identifier		
P	.	*rdf-schema#label*		
R	.	rdf-schema#Literal		

When the file is processed, because E42_identifier has the same source node and is given the same URI it will produce linked triples in RDF, i.e.;

Domain >	**Property >**	**Range / Domain >**	**Property >**	**Range**
E22_Man-Made_Object	P1_is_identified_by	E42_Identifier	rdfs:Label	Literal Value

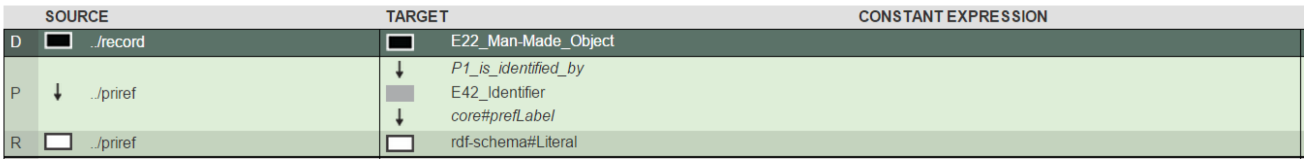
However, to avoid having to create domains when we only have one ultimate target of the mapping we can use an intermediate mapping. This simply means that the ultimate mapping we are interested in goes through an intermediate entity and there are no other mappings we need to address.



SOURCE	TARGET	IF RULE	COMMENTS
D ../record	E22_Man-Made_Object		

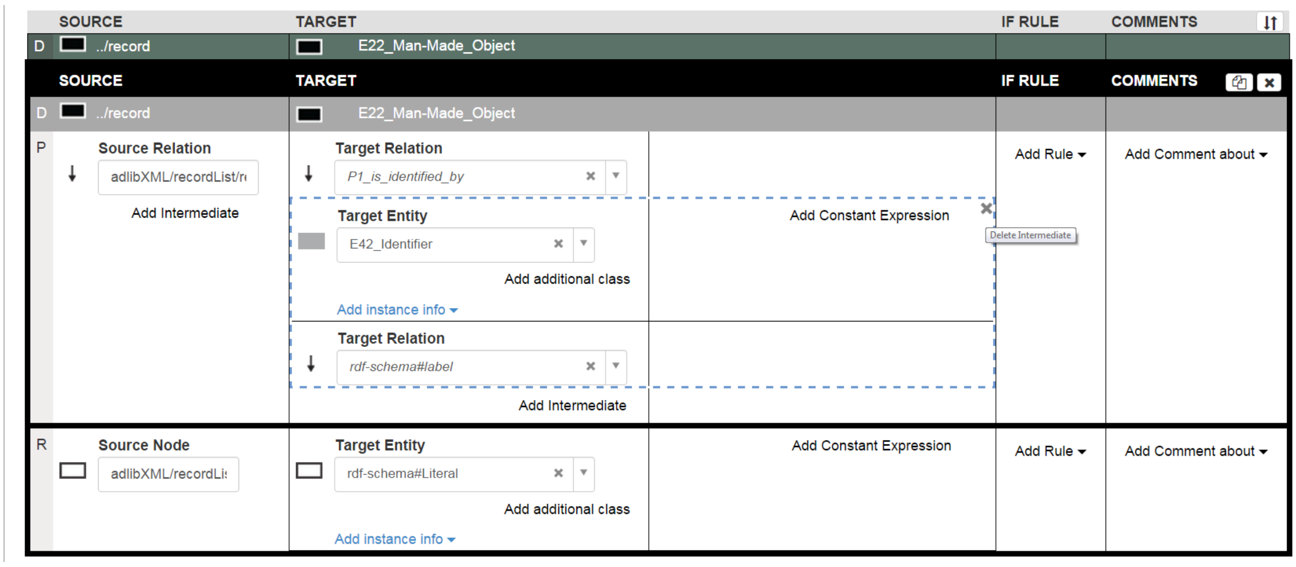
	SOURCE	TARGET		IF RULE	COMMENTS
D	../record	E22_Man-Made_Object			
P	Source Relation adlibXML/recordList/r... Add Intermediate	Target Relation P1_is_identified_by		Add Rule	Add Comment about
		Target Entity E42_Identifier Add additional class Add instance info	Add Constant Expression		
		Target Relation rdf-schema#label Add Intermediate			
R	Source Node adlibXML/record...	Target Entity rdf-schema#Literal Add additional class Add instance info	Add Constant Expression	Add Rule	Add Comment about

In view mode this looks as follows:



	SOURCE	TARGET	CONSTANT EXPRESSION
D	../record	E22_Man-Made_Object	
P	../priref	*P1_is_identified_by* E42_Identifier *core#prefLabel*	
R	../priref	rdf-schema#Literal	

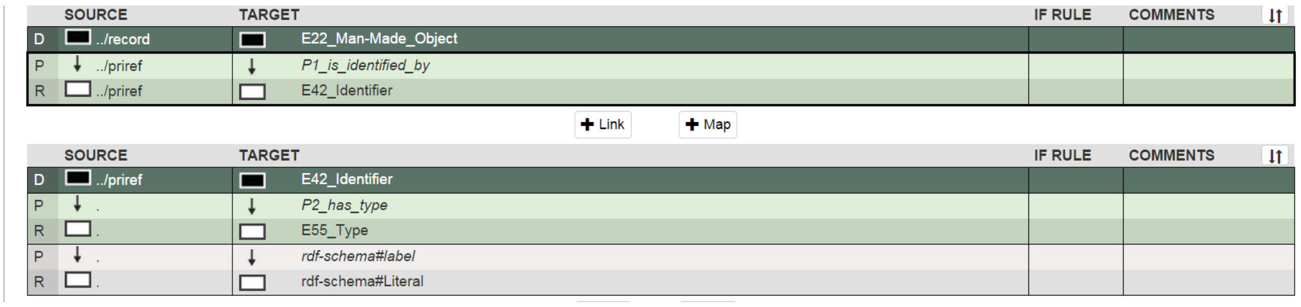
However we also wanted to link to another entity, for example a **Type,** and have a value for Identifier. This means removing the intermediate and replacing it with a separate **Identifier** domain so that we can both document the mapping to the literal identifier value and to the **E55_Type**.



SOURCE	TARGET	IF RULE	COMMENTS
D ../record	E22_Man-Made_Object		

	SOURCE	TARGET		IF RULE	COMMENTS
D	../record	E22_Man-Made_Object			
P	Source Relation adlibXML/recordList/r Add Intermediate	Target Relation P1_is_identified_by		Add Rule	Add Comment about
		Target Entity E42_Identifier Add additional class Add instance info	Add Constant Expression	Delete Intermediate	
		Target Relation rdf-schema#label Add Intermediate			
R	Source Node adlibXML/recordLi	Target Entity rdf-schema#Literal Add additional class Add instance info	Add Constant Expression	Add Rule	Add Comment about

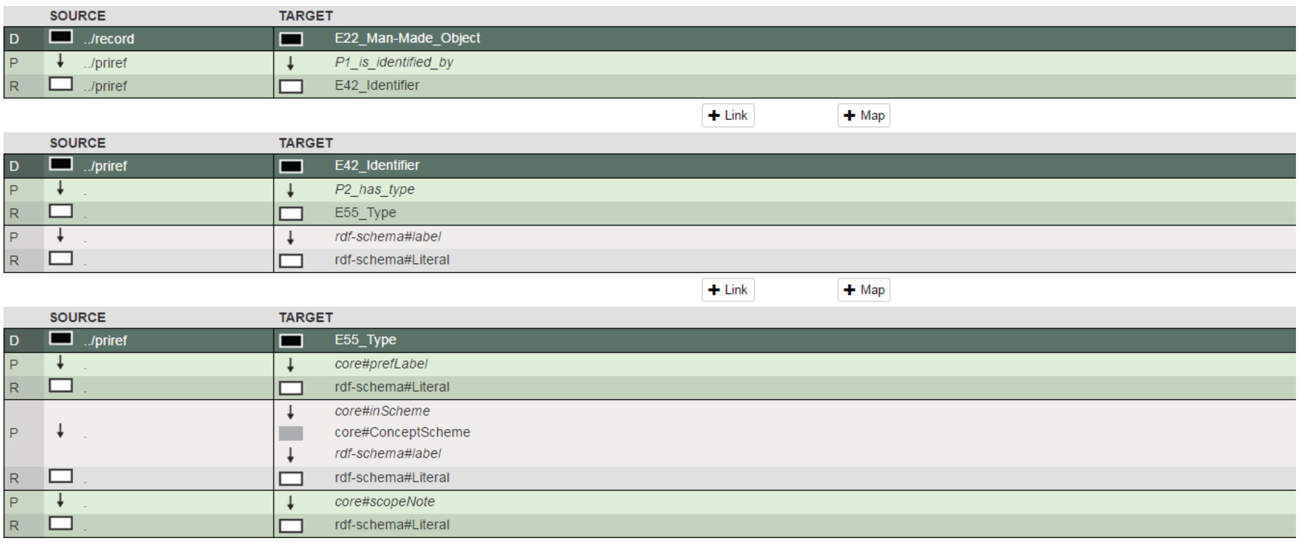
The mapping therefore becomes



	SOURCE	TARGET	IF RULE	COMMENTS
D	../record	E22_Man-Made_Object		
P	../priref	*P1_is_identified_by*		
R	../priref	E42_Identifier		

	SOURCE	TARGET	IF RULE	COMMENTS
D	../priref	E42_Identifier		
P	.	*P2_has_type*		
R	.	E55_Type		
P	.	*rdf-schema#label*		
R	.	rdf-schema#Literal		

This means that **Identifier** has both a label and a link to **Type** and the authority information is added to that Domain.



	SOURCE	TARGET
D	../record	E22_Man-Made_Object
P	../priref	*P1_is_identified_by*
R	../priref	E42_Identifier

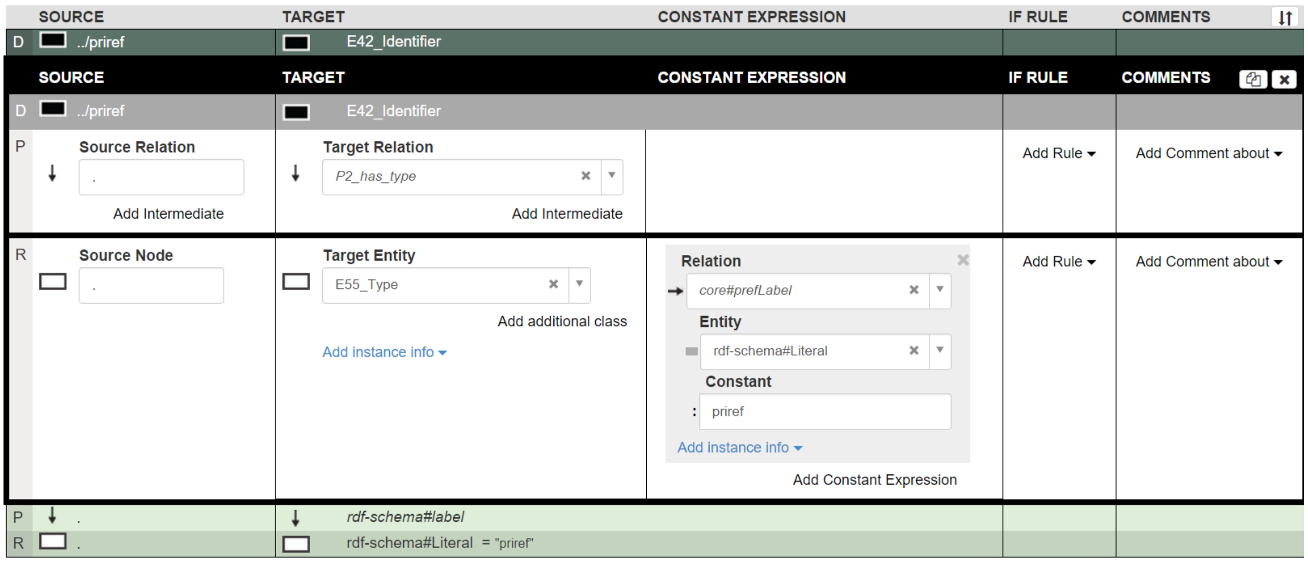
+ Link   + Map

	SOURCE	TARGET
D	../priref	E42_Identifier
P	.	*P2_has_type*
R	.	E55_Type
P	.	*rdf-schema#label*
R	.	rdf-schema#Literal

+ Link   + Map

	SOURCE	TARGET
D	../priref	E55_Type
P	.	*core#prefLabel*
R	.	rdf-schema#Literal
P	.	*core#inScheme* core#ConceptScheme *rdf-schema#label*
R	.	rdf-schema#Literal
P	.	*core#scopeNote*
R	.	rdf-schema#Literal

Alternatively, if the types are simply constants that are the same for the same field then we can use constants in the interface itself (see below). Here a relationship using **skos:prefLabel** uses a literal constant **prefLabel**. With the ability to add literals in the interface itself and the ability to add constants, the number of Domains can be reduced.

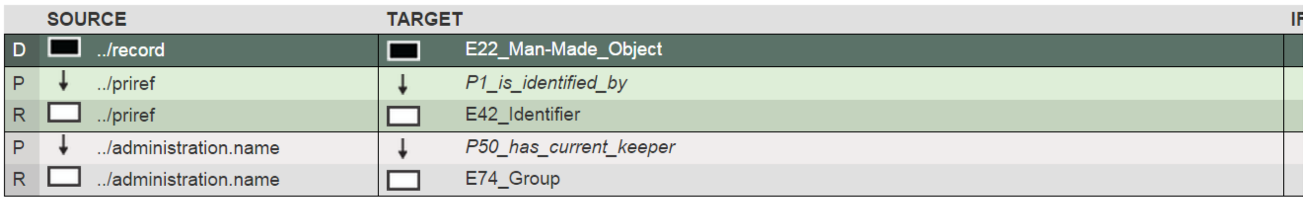


	SOURCE	TARGET	CONSTANT EXPRESSION	IF RULE	COMMENTS
D	../priref	E42_Identifier			
D	../priref	E42_Identifier			
P	Source Relation: . Add Intermediate	Target Relation: *P2_has_type* Add Intermediate		Add Rule	Add Comment about
R	Source Node: .	Target Entity: E55_Type Add additional class Add instance info	Relation: *core#prefLabel* Entity: rdf-schema#Literal Constant: priref Add instance info Add Constant Expression	Add Rule	Add Comment about
P	.	*rdf-schema#label*			
R	.	rdf-schema#Literal = "priref"			

10.3.1 Additional

In the example one field is the basis for a range of different semantic statements. The level of information that is recorded changes the way in which the mapping is asserted. If we were not interested in creating **E55_Type** information for the **E42_Identifier** but want to record a value, then the mapping can be executed within **E22_Man-Made_Object** Domain using E42_**Identifier** as the intermediate. If we had wanted to create an **E55_Type** for the **E42_Identifier** and did not need to add additional information against the **E42_Identifier** then it can be used as an intermediate for **E55_Type**. Because we wanted to record information against both the **E42_Identifie**r entity and the **E55_Type** entity we needed to create separate domains.

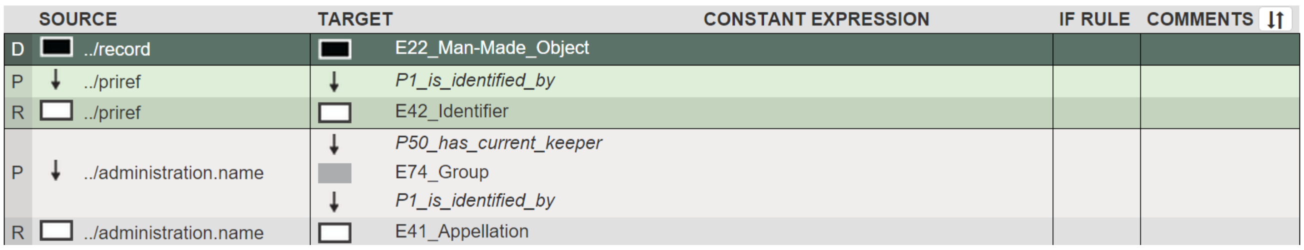
Consider an alternative example using **P1_is_identified_by** with **E41_Appellation. E41_Appellation** is used to name a specific instance of an entity (unlike **E55_Type** which represents concepts and is used to classify and characterize). We may have a department that is the keeper of an object. The mapping would look like this:



		SOURCE		TARGET	IF
D	■	../record	■	E22_Man-Made_Object	
P	↓	../priref	↓	*P1_is_identified_by*	
R	□	../priref	□	E42_Identifier	
P	↓	../administration.name	↓	*P50_has_current_keeper*	
R	□	../administration.name	□	E74_Group	

The object has a department which is recorded in the **Source Node** as **administration.name.** The department is the current keeper of the object. We therefore want to express the relationship between the department and the object and we want to say what the name of the department is. We may therefore want to include an Appellation (name) for the department as a literal.

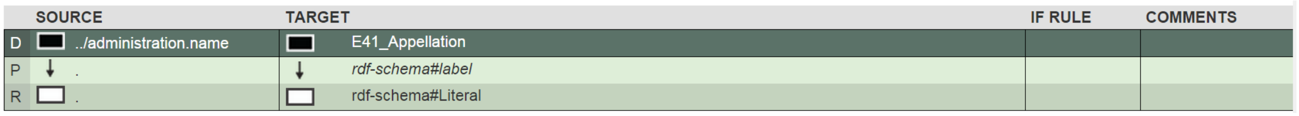
Again this could be expressed through the intermediate of **E39_Group**.



		SOURCE		TARGET	CONSTANT EXPRESSION	IF RULE	COMMENTS
D	■	../record	■	E22_Man-Made_Object			
P	↓	../priref	↓	*P1_is_identified_by*			
R	□	../priref	□	E42_Identifier			
P	↓	../administration.name	↓	*P50_has_current_keeper*			
			▨	E74_Group			
			↓	*P1_is_identified_by*			
R	□	../administration.name	□	E41_Appellation			

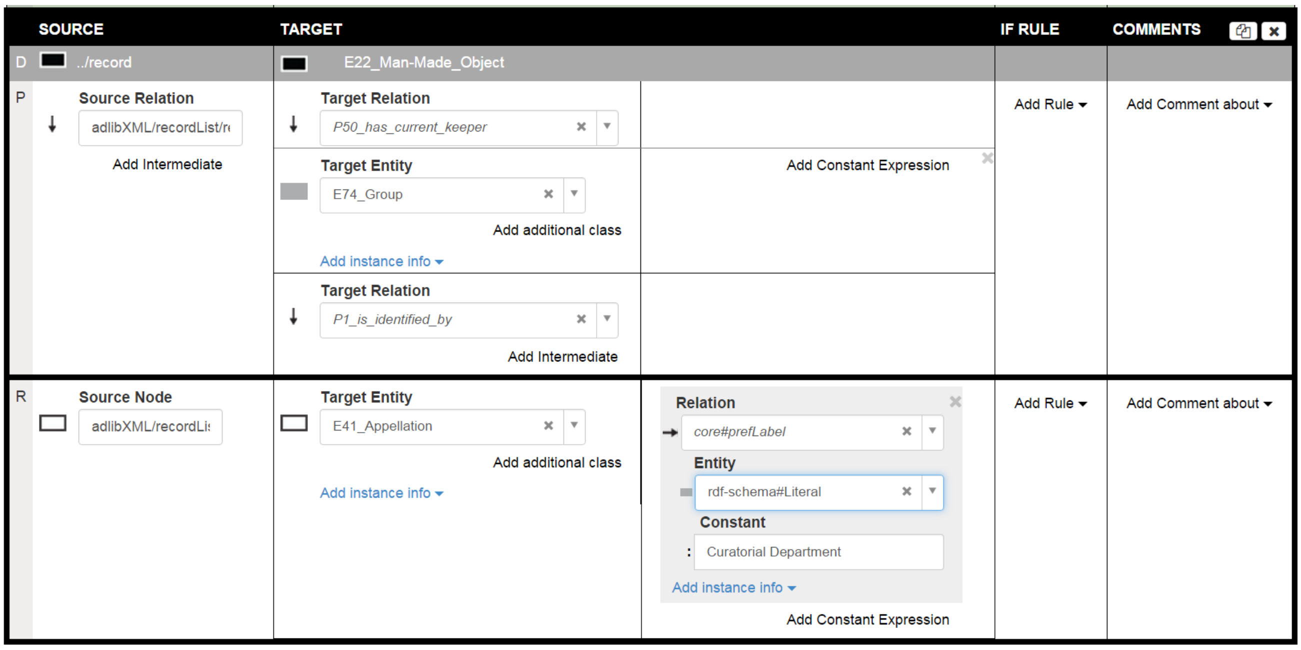
Since it is E41_**Appellation** that provides the entity to attach a label containing the person's name as a string, we can create an intermediate through **E74_Group** to **E41_Appellation.**

The **Appellation** Domain contains a link to a label with the departmental name as a literal.



		SOURCE		TARGET	IF RULE	COMMENTS
D	■	../administration.name	■	E41_Appellation		
P	↓	.	↓	*rdf-schema#label*		
R	□	.	□	rdf-schema#Literal		

However, if we had wanted to describe the type of name with a constant, for example, "curatorial department" (if administrative.name was a field for curatorial departments), then we can add a constant expression for an entity.



	SOURCE	TARGET		IF RULE	COMMENTS
D	■ ../record	■ E22_Man-Made_Object			
P	Source Relation ↓ adlibXML/recordList/r Add Intermediate	Target Relation ↓ *P50_has_current_keeper* Target Entity ▨ E74_Group Add additional class Add instance info ▾ Target Relation ↓ *P1_is_identified_by* Add Intermediate	Add Constant Expression	Add Rule ▾	Add Comment about ▾
R	Source Node □ adlibXML/recordLi	Target Entity □ E41_Appellation Add additional class Add instance info ▾	Relation → *core#prefLabel* Entity rdf-schema#Literal Constant : Curatorial Department Add instance info ▾ Add Constant Expression	Add Rule ▾	Add Comment about ▾

10.4 Multiple Instantiation

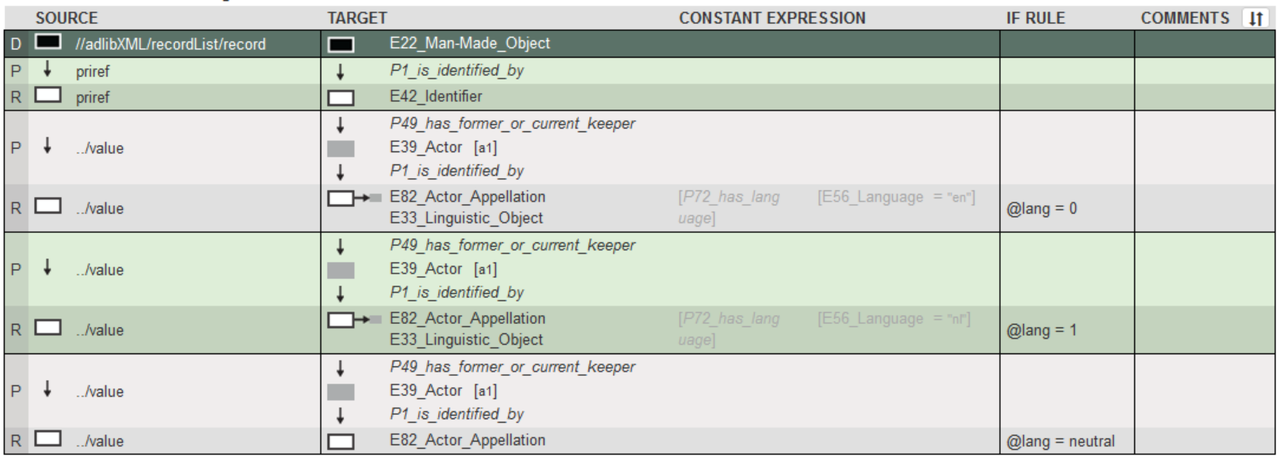
CIDOC CRM supports multiple instantiation. This means that an instance of class A is also regarded as an instance of one or more other classes at the same time. When multiple instantiation is used, it has the effect that all the properties of all these classes become available to describe this instance. In 3M it is possible to define multiple instantiation of a Target Entity by using the **Add additional class** option.

In the following example we use multiple instantiation to encode the language information. Each input record contains the tag <administration.name> which contains a name in three different forms. An example is presented here:

```
<administration.name>
    <value lang="neutral">Nederlandse Geschiedenis</value>
    <value lang="0">History</value>
    <value lang="1">Geschiedenis</value>
</administration.name>
```

The person responsible for the mapping knows that the value 1 for attribute lang represents Dutch while the value 0 represents English.

The administration.name is mapped to an **E39_Actor** who has three different names. Each name is an instance of the **E82_Actor_Appellation** class and at the same time the **E33_Linguistic_Object** class. In this way, the name instance inherits the properties of both the **E82_Actor_Appellation** and **E33_Linguistic_Object** and thus we can assign to each name a constant expression ***P72_has_language -> E56_Language*** with the appropriate value.



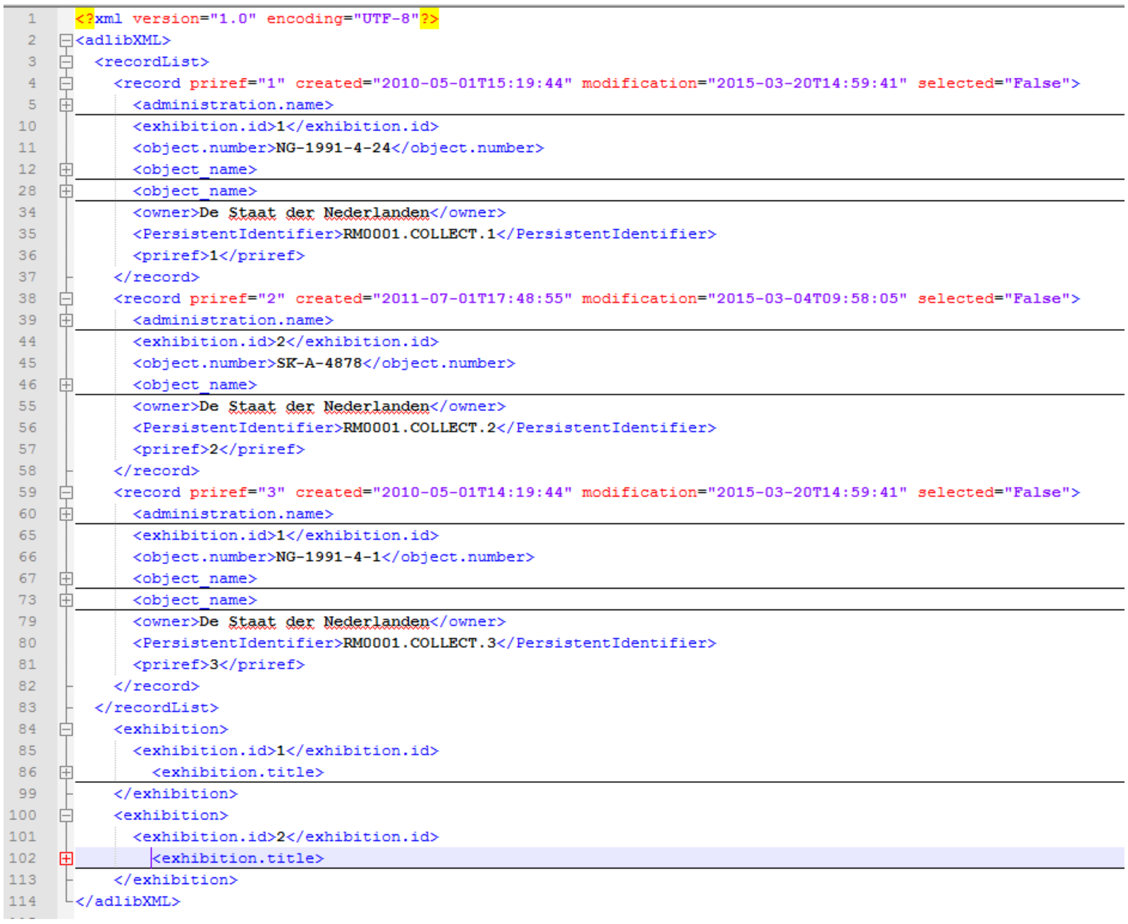
	SOURCE	TARGET	CONSTANT EXPRESSION	IF RULE	COMMENTS
D	//adlibXML/recordList/record	E22_Man-Made_Object			
P	priref	*P1_is_identified_by*			
R	priref	E42_Identifier			
P	../value	*P49_has_former_or_current_keeper* E39_Actor [a1] *P1_is_identified_by*			
R	../value	E82_Actor_Appellation E33_Linguistic_Object	[*P72_has_language*] [E56_Language = "en"]	@lang = 0	
P	../value	*P49_has_former_or_current_keeper* E39_Actor [a1] *P1_is_identified_by*			
R	../value	E82_Actor_Appellation E33_Linguistic_Object	[*P72_has_language*] [E56_Language = "nl"]	@lang = 1	
P	../value	*P49_has_former_or_current_keeper* E39_Actor [a1] *P1_is_identified_by*			
R	../value	E82_Actor_Appellation		@lang = neutral	

Note: In the above example **E39_Actor** is the same Actor in all three Property-Range links. This is declared explicitly by using the **is Same as** option in **Add Instance info** drop down and naming it **a1** (this name is symbolic, it can be anything as long as it is the same in all instances, see also Additional Link Information).

10.5 Relational database JOIN

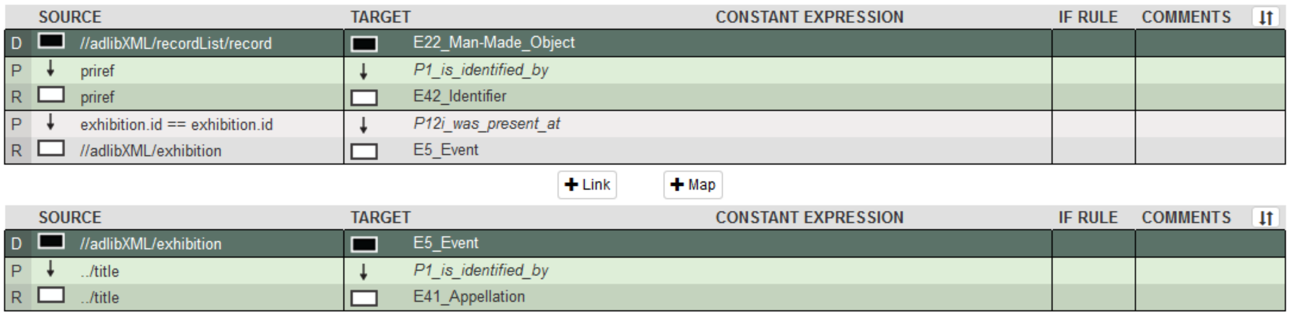
It is quite common that the source XML is the export of a relational data base which uses joins between different relational tables. In such an XML export the different tables are represented by distinct not nested xml tags.

Let's assume that we have two relational tables: **record** and **exhibition**. There is a join between the field **exhibition.id** of table **record** with the **exhibition.id** of the table **exhibition**. The XML output of such a database is displayed below (please note that this example, although based on the Rijksmuseum data, is artificial, created only to demonstrate the join).



```xml
<?xml version="1.0" encoding="UTF-8"?>
<adlibXML>
  <recordList>
    <record priref="1" created="2010-05-01T15:19:44" modification="2015-03-20T14:59:41" selected="False">
      <administration.name>
      <exhibition.id>1</exhibition.id>
      <object.number>NG-1991-4-24</object.number>
      <object_name>
      <object_name>
      <owner>De Staat der Nederlanden</owner>
      <PersistentIdentifier>RM0001.COLLECT.1</PersistentIdentifier>
      <priref>1</priref>
    </record>
    <record priref="2" created="2011-07-01T17:48:55" modification="2015-03-04T09:58:05" selected="False">
      <administration.name>
      <exhibition.id>2</exhibition.id>
      <object.number>SK-A-4878</object.number>
      <object_name>
      <owner>De Staat der Nederlanden</owner>
      <PersistentIdentifier>RM0001.COLLECT.2</PersistentIdentifier>
      <priref>2</priref>
    </record>
    <record priref="3" created="2010-05-01T14:19:44" modification="2015-03-20T14:59:41" selected="False">
      <administration.name>
      <exhibition.id>1</exhibition.id>
      <object.number>NG-1991-4-1</object.number>
      <object_name>
      <object_name>
      <owner>De Staat der Nederlanden</owner>
      <PersistentIdentifier>RM0001.COLLECT.3</PersistentIdentifier>
      <priref>3</priref>
    </record>
  </recordList>
    <exhibition>
      <exhibition.id>1</exhibition.id>
        <exhibition.title>
    </exhibition>
    <exhibition>
      <exhibition.id>2</exhibition.id>
        <exhibition.title>
    </exhibition>
</adlibXML>
```

In X3ML we use the special **join** operator (==) in the source path to denote relational database joins. The left part of the join is evaluated relative to the source Domain (//adlibXML/recordList/record) while the right part of the join is evaluated relative to the Range (//adlibXML/exhibition). The Range will also be the Domain in a separate map. The X3ML engine is responsible to create the URIs in such a way that the **E5_Event** (target Range in the first map) is the same with the **E5_Event** (target Domain in the second map).

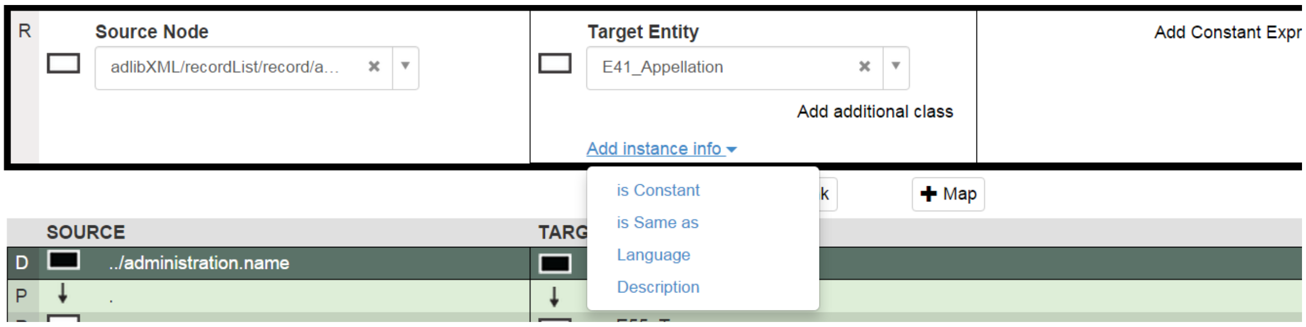


	SOURCE	TARGET	CONSTANT EXPRESSION	IF RULE	COMMENTS
D	■ //adlibXML/recordList/record	■ E22_Man-Made_Object			
P	↓ priref	↓ *P1_is_identified_by*			
R	□ priref	□ E42_Identifier			
P	↓ exhibition.id == exhibition.id	↓ *P12i_was_present_at*			
R	□ //adlibXML/exhibition	□ E5_Event			

	SOURCE	TARGET	CONSTANT EXPRESSION	IF RULE	COMMENTS
D	■ //adlibXML/exhibition	■ E5_Event			
P	↓ ../title	↓ *P1_is_identified_by*			
R	□ ../title	□ E41_Appellation			

10.6 Additional Link Information

Each mapping allows the insertion of additional instance information.



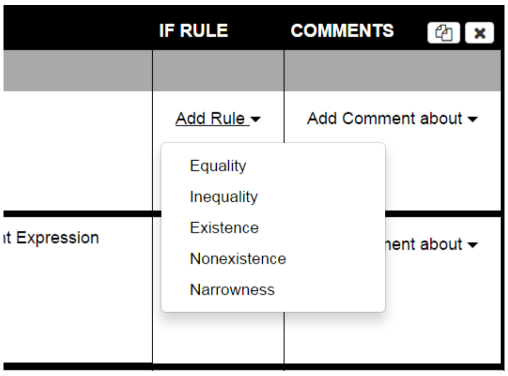
is Constant - allows a constant to be defined for the particular entity which will be taken into account during the definition of the instance generator for this entity. It actually instructs the URI expert who is responsible for the instance generation functions to use a constant value in the instance and label generators of this particular entity.

is Same as - allows the insertion of a label to use when you want to tell 3M that two instances used in a number of locations in the same mapping table are the same. For example, if you use an **E12 Production** as an intermediate node in many paths in your mapping and you want to ensure that 3M knows that it is the same **E12 Production** in every case, simply give each **E12 Production** the same ‘is Same as’ label. This works within the same domain.

Language – Allows a language code to be inserted to denote the language being used. For example, a literal may be using a particular language. As in the “is Constant” case, the Language information is going to be used by the person responsible for the instance and label generation.

Description – General Information to help the person employed to create the URI the policy.

10.7 Links and Rules


IF RULE
COMMENTS
Add Rule
Add Comment about
Equality
Inequality
Existence
Nonexistence
Narrowness
t Expression
ent about

3M allows certain rules to be defined. These are rules about the information that is contained in the Source Nodes. These don't necessarily have to be the Source Node for the current mapping. They could be rules that use information from other Source Nodes, using XPath notation.

Currently an IF RULE applies to all the Property-Range tuple. This means that if the IF RULE is evaluated to false, nothing is created i.e. no target link and no target entity. So it doesn't really make any difference if the IF RULE is defined in the property or in the range. The end result will be the same. If there is an IF RULE defined both in the property and in the range it is as if having an AND between the two rules.

Currently there is no "else" capability support, so the only way to define the else is by duplicating the Property-Range tuple and changing the IF RULE accordingly.

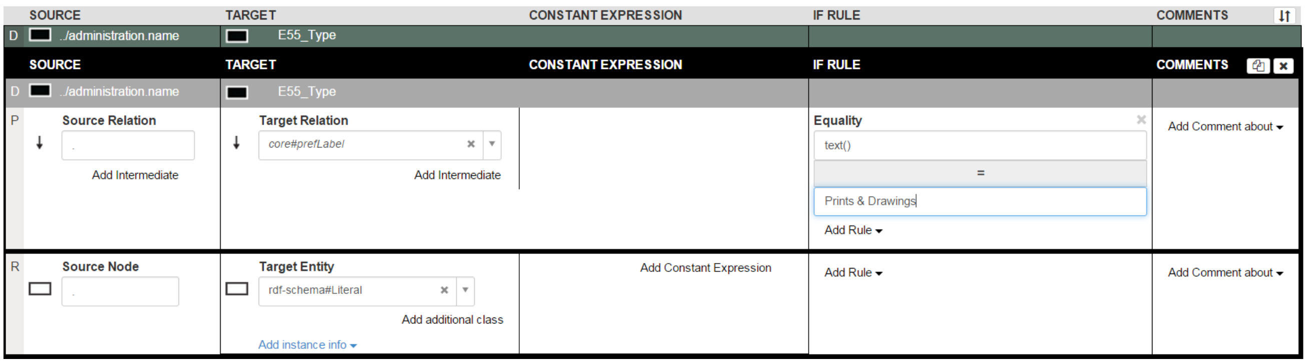
One of the reasons that IF RULEs are allowed both in the property and the range is to indicate where the value that is checked is coming from. But since we deal with XPaths, rules may use information from other Source Nodes too. Additionally we have the option to expand the design later on in order to support a better "else" mechanism.

10.7.1 Equality and Inequality

This allows 'If' based statements. If they evaluate 'True' then the mapping for the current instance is executed. For example;

10.7.2 Testing the value of the current element:

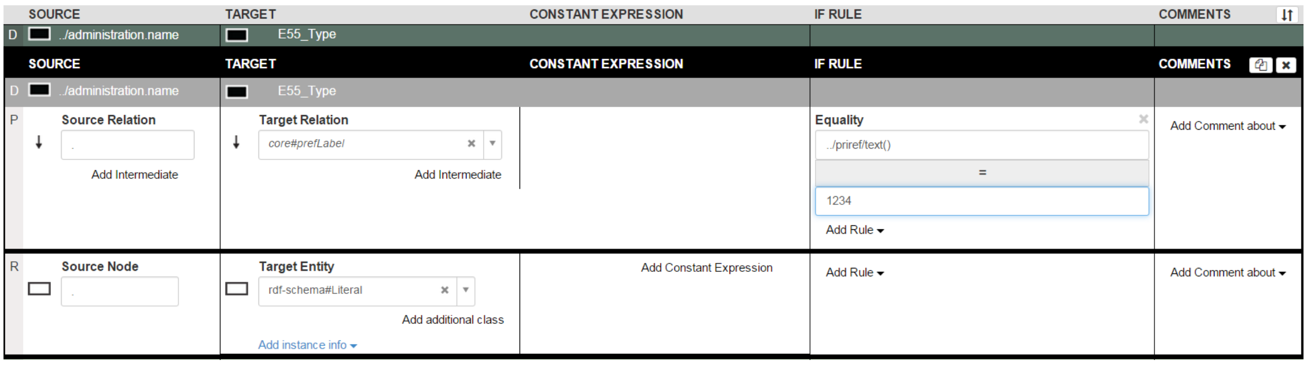
If text() = "Prints & Drawing"


SOURCE | TARGET | CONSTANT EXPRESSION | IF RULE | COMMENTS
D ./administration.name | E55_Type
SOURCE | TARGET | CONSTANT EXPRESSION | IF RULE | COMMENTS
D ./administration.name | E55_Type
P Source Relation | Target Relation core#prefLabel | Equality text() = Prints & Drawings Add Rule | Add Comment about
Add Intermediate | Add Intermediate
R Source Node | Target Entity rdf-schema#Literal | Add Constant Expression | Add Rule | Add Comment about
Add additional class
Add instance info

10.7.3 Testing another element

On the same level as the current Source Node element

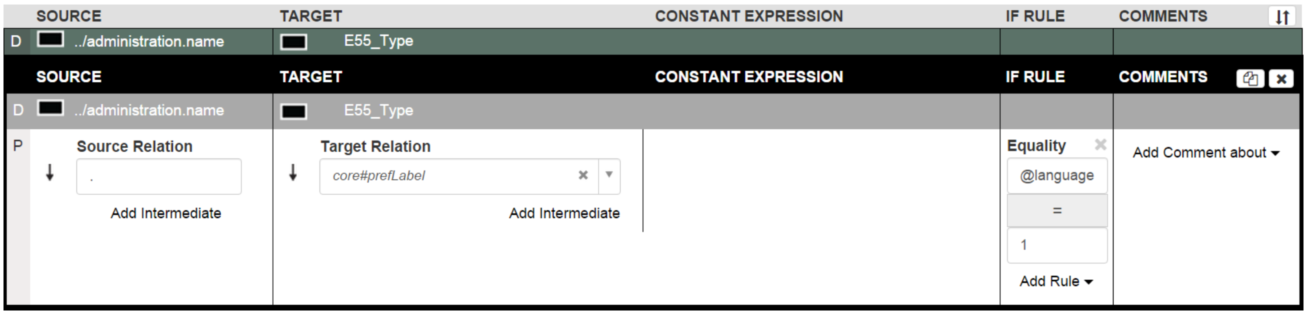
If ../priref/text() = 1234


SOURCE | TARGET | CONSTANT EXPRESSION | IF RULE | COMMENTS
D ./administration.name | E55_Type
SOURCE | TARGET | CONSTANT EXPRESSION | IF RULE | COMMENTS
D ./administration.name | E55_Type
P Source Relation | Target Relation core#prefLabel | Equality ../priref/text() = 1234 Add Rule | Add Comment about
Add Intermediate | Add Intermediate
R Source Node | Target Entity rdf-schema#Literal | Add Constant Expression | Add Rule | Add Comment about
Add additional class
Add instance info

10.7.4 Testing the Current Attribute

Where the attribute is @language = “1”, @language = “2”, and so on.

If @language = 1

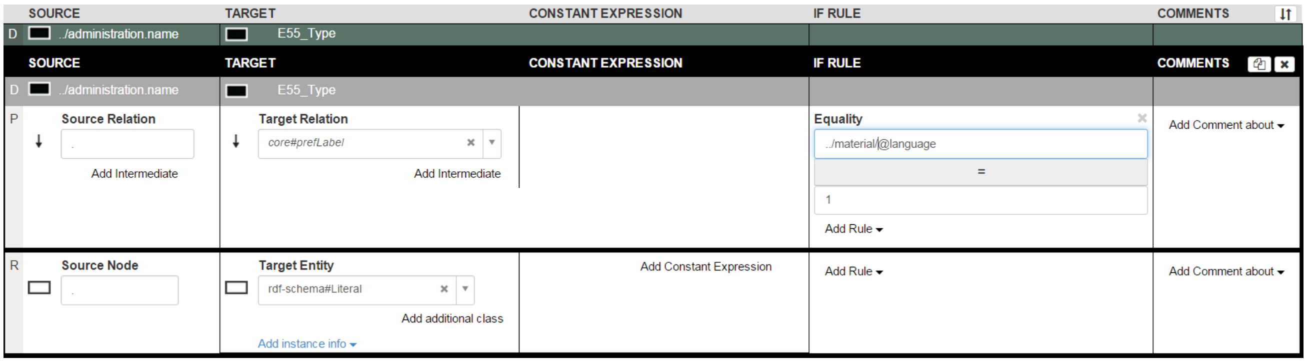


SOURCE	TARGET	CONSTANT EXPRESSION	IF RULE	COMMENTS
D ../administration.name	E55_Type			
P Source Relation: . / Add Intermediate	Target Relation: core#prefLabel / Add Intermediate		Equality: @language = 1 / Add Rule	Add Comment about

10.7.5 Testing the attribute of another Element

Where the element is on the same level as the current Source Node where the element is called material and the attribute is @language = “1”, @language = “2”, and so on.

If ../material/@language = 1



SOURCE	TARGET	CONSTANT EXPRESSION	IF RULE	COMMENTS
D ../administration.name	E55_Type			
P Source Relation: . / Add Intermediate	Target Relation: core#prefLabel / Add Intermediate		Equality: ../material/@language = 1 / Add Rule	Add Comment about
R Source Node: .	Target Entity: rdf-schema#Literal / Add additional class / Add instance info	Add Constant Expression	Add Rule	Add Comment about

10.7.6 InEquality

This is the same as Equality except that you are testing a negative.

10.8 Existence & Non-Existence

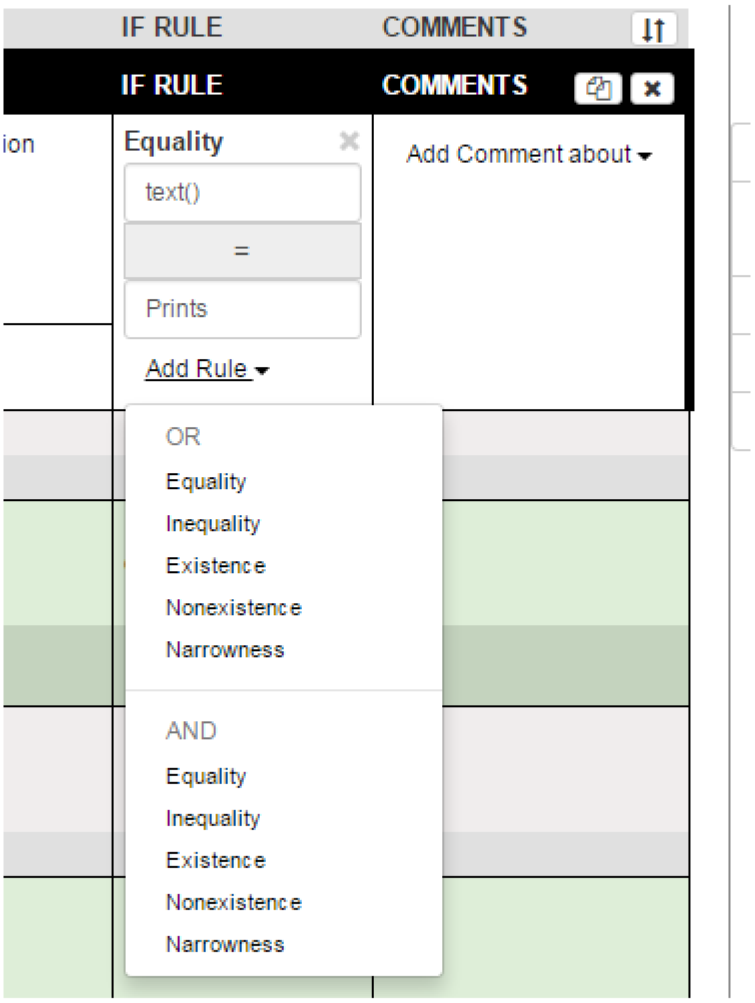
This simply tests that a Source Node actually exists. Its purpose is to ensure that transformation isn't halted simply because a fields doesn't appear in a particular record. This should be used carefully and excessive use might indicate issues with the data. Non-Existence tests for the opposite.

10.9 Narrowness

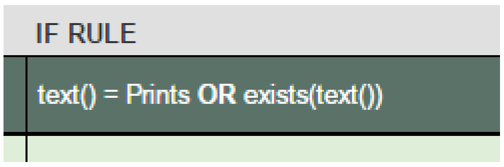
The mapping is only executed against a test that one concept term is narrower than another.

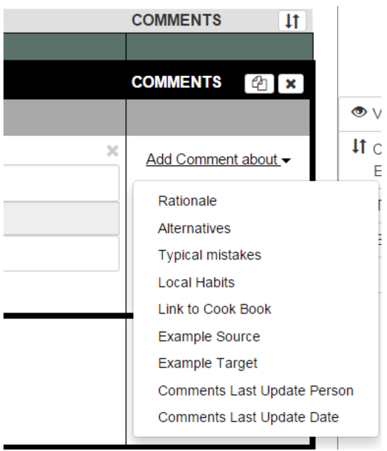
10.10 Multiple rules

Sometimes more than one rules have to be defined. 3M allows combining multiple rules using OR and AND operators.



If OR Existence is selected and value provided is text(), then you get the following result (in View mode):





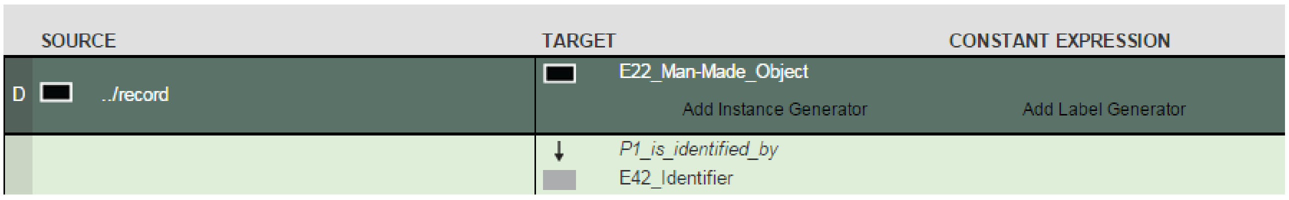
10.11 Comments

The comments section allows mappers to write useful comments for exchanging information about mapping different features with the community. These are divided into different types of comment with the intention that such comments could be compiled into a mapping knowledgebase.

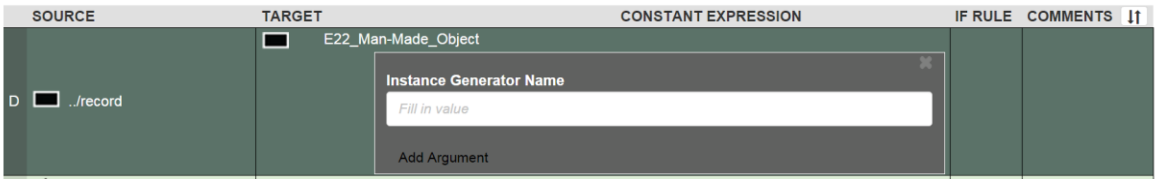
11 Instance and Label Generation

The instance and label generator editor is located on the **More** menu and is labelled as “Generators”. This is the tool that creates Uniform Resource Indicators and Labels for the mapped data which is assigned as the source data and transformed by the X3ML mapping engine. If we take the earlier example, the XML Source Node, **adlibXML/recordList/record** is mapped to **E22_Man-Made_Object.** The instances of this mapping need to be assigned a unique URI and labels.

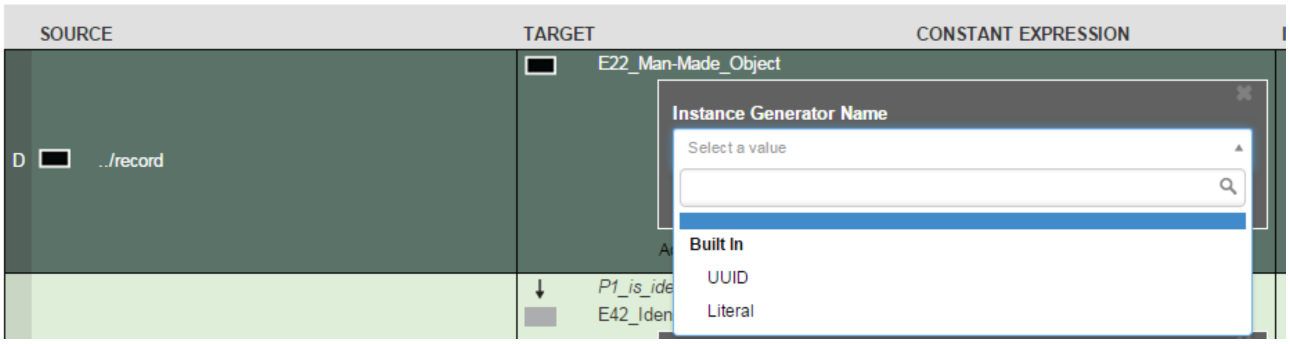
The Generators editor replicates the mapping table but prompts for an Instance Generator and Label Generators.



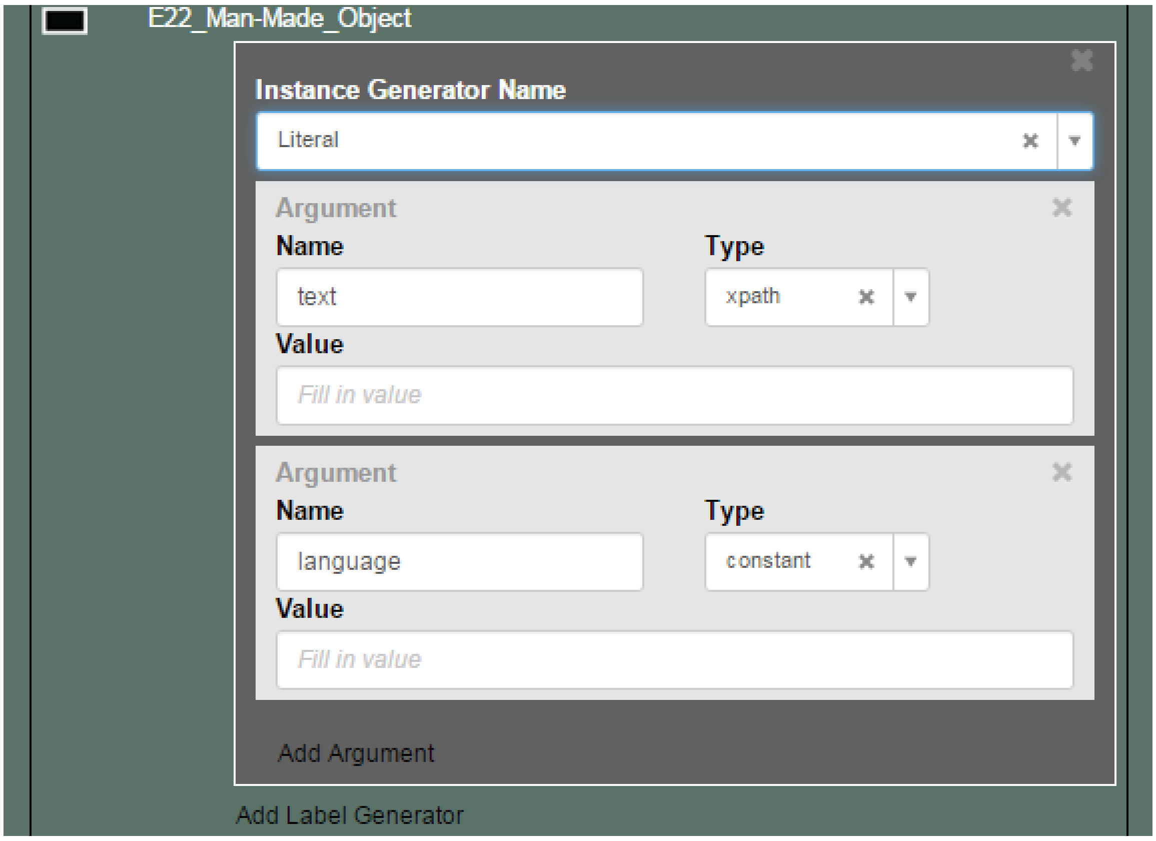
Clicking on an Instance generator link produces the following screen promoting for a Generator Name (If option “Generators” in Configuration tab is set to “Manual”). Once you fill in the name, you may add as many arguments as you want.



On the other hand, if option “Generators” is set to “Auto” in the Configuration tab, the following screen is produced:



Once you choose an instance generator name, appropriate arguments are fetched automatically and you simply have to fill in the values.


E22_Man-Made_Object

Instance Generator Name: Literal

Argument
Name: text
Type: xpath
Value: Fill in value

Argument
Name: language
Type: constant
Value: Fill in value

Generators are templates for creating a URI and the same generator can be used throughout the mapping if it is applicable. The templates need to be defined separately and then uploaded to the mapping (using the generator policy field in Info tab), in order to be used by the generators. The templates are created in XML and look like this;

```
<generator_policy>

<generator name="PersistentIdThing" prefix="rijks">

        <pattern>{id}/{thing}/{identifier}</pattern>

        </generator>

<generator_policy>
```

This simply means that there is a generator called **PersistentIdThing** and the URI pattern has two elements.

1. The namespace. This is provided by the prefix. You will remember from earlier in this document that a prefix is a shorter version of a full namespace that needs to be configured in the **Info** tab of the mapping. Therefore if we configured the prefix **rijks** for the full namespace http://collection.rijksmuseum.org/ then this is the first part of the URI that will be generated by this pattern
2. The rest of the pattern is then defined by the pattern inserted between the pattern element tags.

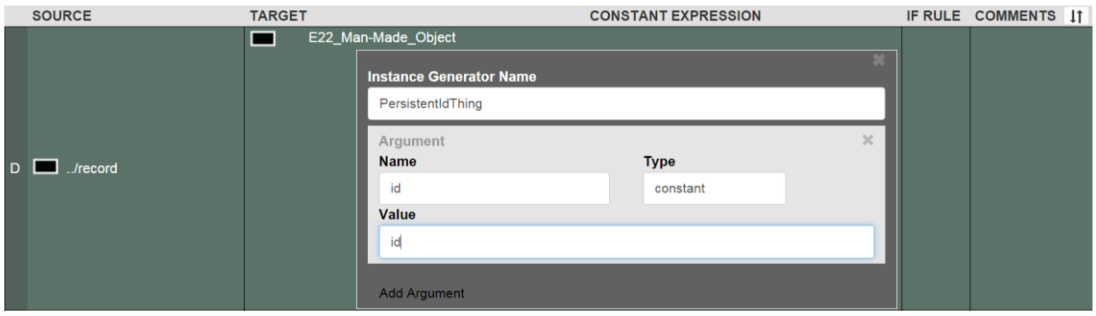
Currently the URI for this template is http://collection.rijksmuseum.org/{id}/{thing}/{identifier}

We haven't yet defined what **{id}, {thing} and {identifier}** are. That's what the Instance generator Editor is for!

For the Instance Generator Name we therefore can use this template name and then we have two choices:

- If option "Generators" in Configuration tab is set to "Auto" and a generator policy file containing **PersistentIdThing** is uploaded , arguments are filled in automatically as soon as we choose generator name **PersistentIdThing**. User only has to fill in values.

- If option "Generators" in Configuration tab is set to "Manual", we can "**Add Argument"** for each of the three variables to define exactly what these will be replaced with.


SOURCE | TARGET | CONSTANT EXPRESSION | IF RULE | COMMENTS

D ../record

E22_Man-Made_Object

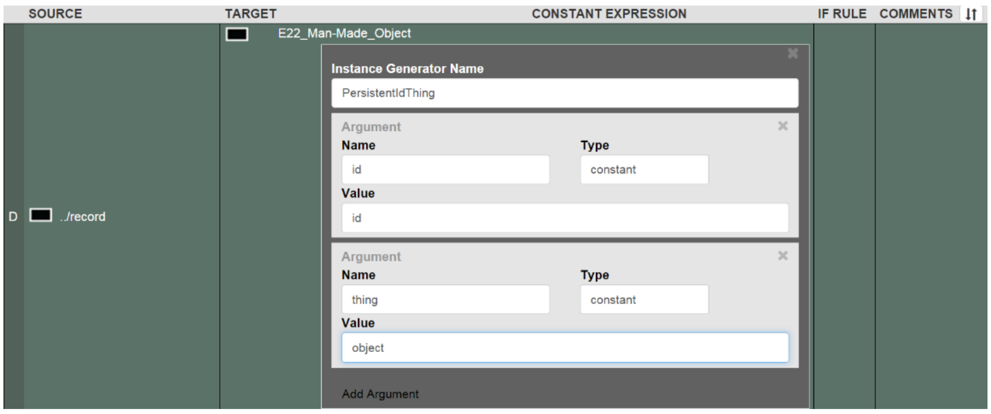
Instance Generator Name: PersistentIdThing

Argument
Name: id
Type: constant
Value: id

Add Argument

In this first part of the generator the name **PersistentIdThing** is added and the first argument {id} is defined. In this example, {id} (the pattern variable name we defined in the XML file) is typed as a **constant** (a hardcoded literal value) which has the value - id. Therefore the URI now becomes;

http://collection.rijksmuseum.org/id/{thing}/{identifier}


SOURCE | TARGET | CONSTANT EXPRESSION | IF RULE | COMMENTS

D ../record

E22_Man-Made_Object

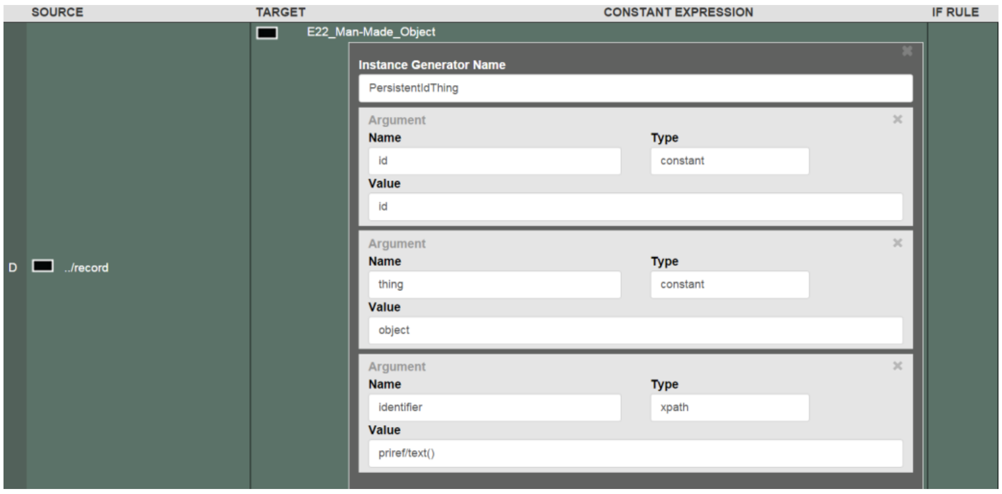
Instance Generator Name: PersistentIdThing

Argument
Name: id
Type: constant
Value: id

Argument
Name: thing
Type: constant
Value: object

Add Argument

The second argument deals with the second variable in the pattern. **{thing}** is replaced by another constant - object


SOURCE | TARGET | CONSTANT EXPRESSION | IF RULE

D ../record

E22_Man-Made_Object

Instance Generator Name: PersistentIdThing

Argument
Name: id
Type: constant
Value: id

Argument
Name: thing
Type: constant
Value: object

Argument
Name: identifier
Type: xpath
Value: priref/text()

The last variable is replaced not by a constant value but a value in the source XML. It's the actual object id from the field, priref. The type is xpath which expects an XPath statement. Since the document is currently at <record> and <priref> is a child element of <record> we can simply use priref/text() to extract the value we want. The result is (if priref had the value of 12 for this particular instance;

http://collection.rijksmuseum.org/id/object/12

The full set of types are:

constant:	a literal hardcoded value
xpath:	an xpath statement to extract some data from the source XML
position:	a counter. If value is left blank it starts from 1.

11.1 Custom Instance Generators

As described already, instance generators are templates that the user can define according to the needs of the data that are being transformed. However there are instance generators that are more complex and cannot be defined through a template. These are the Custom Instance Generators and are implemented via code. X3ML allows the incorporation of new custom instance generators. The Custom Instance Generators that are currently implemented are:

11.1.1 UUID

This function creates a UUID. It does not need to be defined in the generator policy file. It is built in the X3ML engine.
Usage in the X3ML file:

```
<instance_generator name="UUID"/>
```

11.1.2 Literal

This function creates a Literal. It does not need to be defined in the generator policy file. It is built in the X3ML engine.
Usage in the X3ML file:

```
<instance_generator name="Literal">
        <arg name="text">text()</arg>
        <arg name="language" type="constant">de</arg>
</instance_generator>
```

11.1.3 URIorUUID

This function checks the argument “text” and if it is a valid URI it copies it to the rdf output otherwise it creates a UUID.
Definition in the Generator Policy file:

```
<generator name="URIorUUID">
        <custom generatorClass="gr.forth.URIorUUID">
                <set-arg name="text"/>
        </custom>
</generator>
```

Usage in the X3ML file:

```
<instance_generator name="URIorUUID">
        <arg name="text">text()</arg>
</instance_generator>
```

11.1.4 Dates

Currently the two functions GermanDateTime and BMdates are the same.

Definition in the Generator Policy file:

```
<generator name="GermanDateTime">
        <custom generatorClass="gr.forth.GermanDate">
                <set-arg name="bound" type="constant"/>
                <set-arg name="text"/>
        </custom>
</generator>
```

Definition in the Generator Policy file:

```
<generator name="BMdates">
        <custom generatorClass="gr.forth.GermanDate">
                <set-arg name="bound" type="constant"/>
```

```
                    <set-arg name="text"/>
            </custom>
    </generator>
```

Usage in the X3ML file:

```
    <instance_generator name="BMdates">
            <arg name="bound">Upper</arg>
            <arg name="text">text()</arg>
    </instance_generator>

    <instance_generator name="GermanDateTime">
            <arg name="bound">Lower</arg>
            <arg name="text">text()</arg>
    </instance_generator>
```

11.2 Label Generators

X3ML handles label generators in a similar way to instance generators. Each created instance can be assigned one instance generator but more than one label generators.

There are two built in label generators: Literal which will create an rdfs:label and prefLabel which will create a skos:prefLabel.

Usage in the X3ML file:

```
    <label_generator name="Literal">

        <arg name="text" type="xpath">text()</arg

    </label_generator>



    <label_generator name="prefLabel">

        <arg name="text" type="xpath">text()</arg

    </label_generator>
```

Label generators can also be defined with templates as instance generators:

Definition in the Generator Policy file:

```
    <generator name="SimpleLabel">
            <pattern>{label}</pattern>
    </generator>
```

Usage in the X3ML file:

```
    <label_generator name="SimpleLabel">
            <arg name="label">text()</arg>
    </label_generator>

    <label_generator name="SimpleLabel">
            <arg name="label" type="constant">The-British-Museum</arg>
    </label_generator>

    <label_generator name="SimpleLabel">
            <arg name="label" type="constant">The-British-Museum</arg>
            <arg name="language" type="constant">en</arg>
    </label_generator>
```

A typical range in X3ML would look like:

```
….
<range>
        <source_node>priref</source_node>
        <target_node>
          <entity>
            <type>crm:E42_Identifier</type>
            <instance_generator name="PersistentIdThing">
              <arg name="persistent" type="xpath">../PersistentIdentifier/text()</arg>
              <arg name="thing" type="constant">priref</arg>
            </instance_generator>
           <label_generator name="Literal">
              <arg name="text" type="xpath">../../PersistentIdentifier/text()</arg>
            </label_generator>
          </entity>
        </target_node>
</range>
```

For the objects with:

```
        <record priref="1" >
        <PersistentIdentifier>RM0001.COLLECT.1</PersistentIdentifier>
        </record>
        <record priref="2" >
        <PersistentIdentifier>RM0001.COLLECT.2</PersistentIdentifier>
        </record>
```

and generator:

```
        xmlns:han=http://hdl.handle.net/10934/
         <generator name="PersistentIdThing" prefix="han">
                <pattern>{persistent}/{thing}</pattern>
         </generator>
```

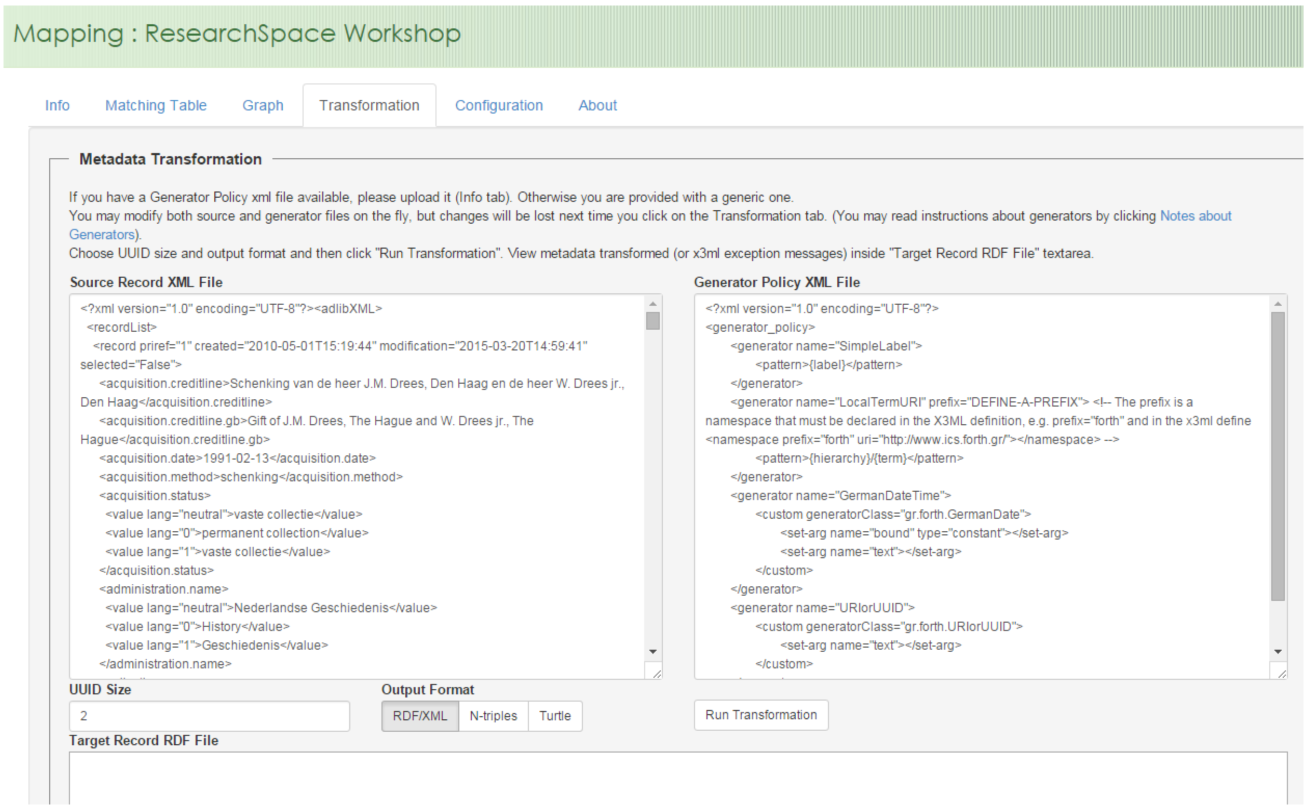
will produce the rdf triples:

```
        <crm:E42_Identifier rdf:about="http://hdl.handle.net/10934/RM0001.COLLECT.1/priref"/>
                <rdfs:label> RM0001.COLLECT.1</rdfs:label>
        </crm:E42_Identifier>
        <crm:E42_Identifier rdf:about="http://hdl.handle.net/10934/RM0001.COLLECT.2/priref"/>
                <rdfs:label> RM0001.COLLECT.2</rdfs:label>
        </crm:E42_Identifier>
```

12 Transformation

When a mapping is complete it should consist of a mapping file (X3ML), a source record file (XML) and a generator policy file (XML). Both XML files should have been uploaded using field in Info tab. If a source record file exists, then transformation to RDF is possible and Transformation tab is enabled.

Once you click on it, following screen is produced:



Metadata Transformation

If you have a Generator Policy xml file available, please upload it (Info tab). Otherwise you are provided with a generic one.
You may modify both source and generator files on the fly, but changes will be lost next time you click on the Transformation tab. (You may read instructions about generators by clicking Notes about Generators).
Choose UUID size and output format and then click "Run Transformation". View metadata transformed (or x3ml exception messages) inside "Target Record RDF File" textarea.

Source Record XML File

```
<?xml version="1.0" encoding="UTF-8"?><adlibXML>
 <recordList>
  <record priref="1" created="2010-05-01T15:19:44" modification="2015-03-20T14:59:41"
selected="False">
   <acquisition.creditline>Schenking van de heer J.M. Drees, Den Haag en de heer W. Drees jr.,
Den Haag</acquisition.creditline>
   <acquisition.creditline.gb>Gift of J.M. Drees, The Hague and W. Drees jr., The
Hague</acquisition.creditline.gb>
   <acquisition.date>1991-02-13</acquisition.date>
   <acquisition.method>schenking</acquisition.method>
   <acquisition.status>
    <value lang="neutral">vaste collectie</value>
    <value lang="0">permanent collection</value>
    <value lang="1">vaste collectie</value>
   </acquisition.status>
   <administration.name>
    <value lang="neutral">Nederlandse Geschiedenis</value>
    <value lang="0">History</value>
    <value lang="1">Geschiedenis</value>
   </administration.name>
```

Generator Policy XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<generator_policy>
    <generator name="SimpleLabel">
        <pattern>{label}</pattern>
    </generator>
    <generator name="LocalTermURI" prefix="DEFINE-A-PREFIX"> <!-- The prefix is a
namespace that must be declared in the X3ML definition, e.g. prefix="forth" and in the x3ml define
<namespace prefix="forth" uri="http://www.ics.forth.gr/"></namespace> -->
        <pattern>{hierarchy}/{term}</pattern>
    </generator>
    <generator name="GermanDateTime">
        <custom generatorClass="gr.forth.GermanDate">
            <set-arg name="bound" type="constant"></set-arg>
            <set-arg name="text"></set-arg>
        </custom>
    </generator>
    <generator name="URIorUUID">
        <custom generatorClass="gr.forth.URIorUUID">
            <set-arg name="text"></set-arg>
        </custom>
```

UUID Size: 2

Output Format: [✓ RDF/XML] N-triples Turtle

Run Transformation

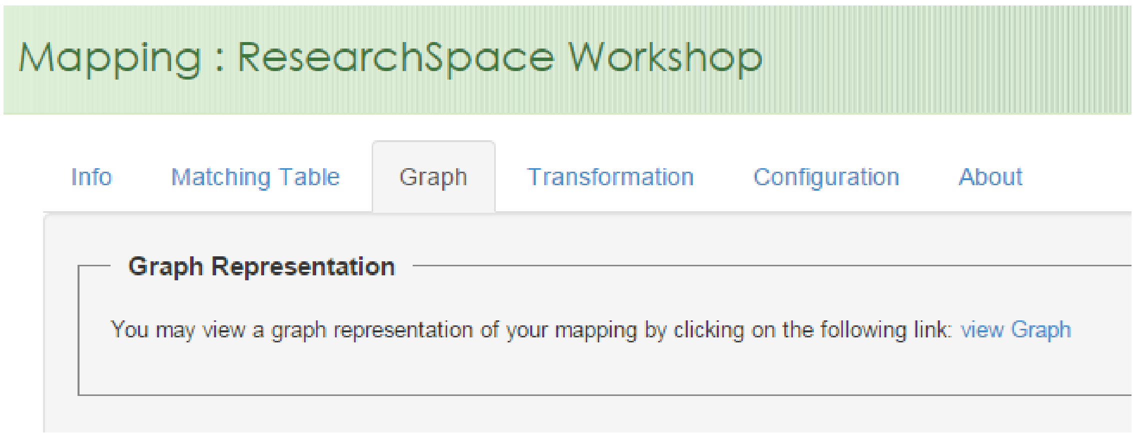
Target Record RDF File

Source Record XML File and Generator Policy XML File panes are filled in using uploaded XML files. If a generator policy file is not available, then a generic one is used (Not recommended). You may edit both panes, to test things on the fly, but keep in mind that changes are not saved.

Choose the type of RDF output you want (RDF/XML, N-triples or Turtle) and desired UUID Size and then run the transformation to get result in Target Record RDF File pane.

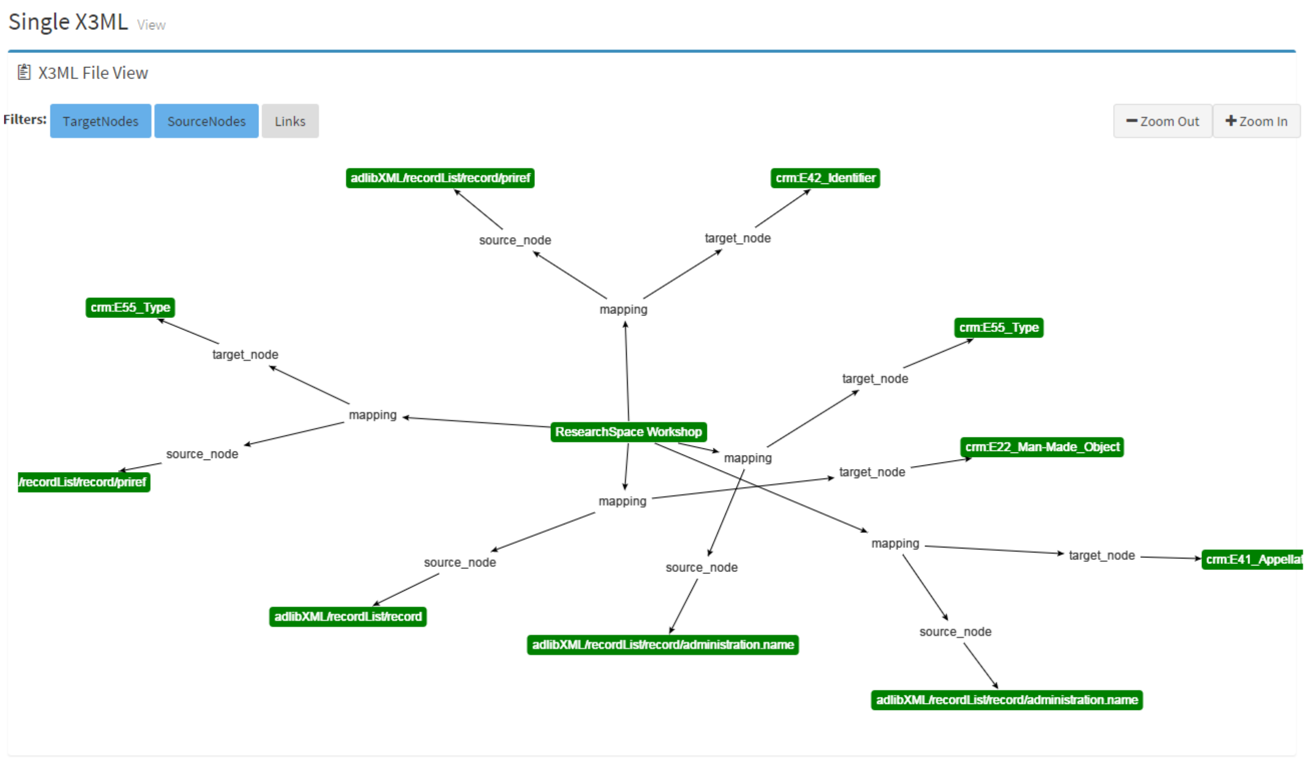
13 Graph

You may view a graph representation of your mapping by using the Graph tab and clicking the “view Graph” link:

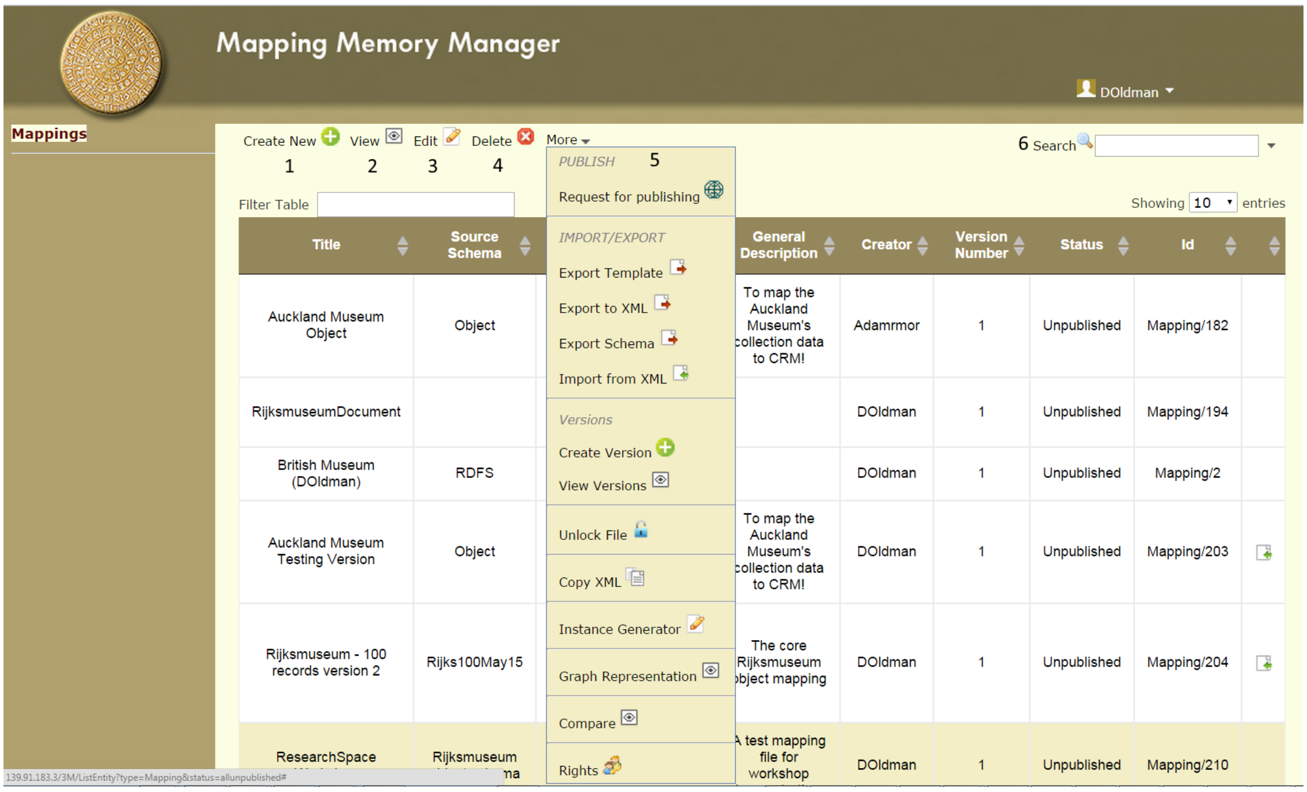


Graph Representation

You may view a graph representation of your mapping by clicking on the following link: view Graph


Single X3ML View
X3ML File View
Filters:
TargetNodes
SourceNodes
Links
Zoom Out
Zoom In
adlibXML/recordList/record/priref
crm:E42_Identifier
source_node
target_node
mapping
crm:E55_Type
target_node
crm:E55_Type
target_node
mapping
ResearchSpace Workshop
mapping
crm:E22_Man-Made_Object
source_node
/recordList/record/priref
mapping
target_node
mapping
target_node
crm:E41_Appellat
source_node
source_node
adlibXML/recordList/record
source_node
adlibXML/recordList/record/administration.name
adlibXML/recordList/record/administration.name

14 Quick Reference to Other 3M functions



Mapping Memory Manager

DOldman

Mappings

Create New 1 | View 2 | Edit 3 | Delete 4 | More 5 | 6 Search

Filter Table

Showing 10 entries

More:
- PUBLISH
  - Request for publishing
- IMPORT/EXPORT
  - Export Template
  - Export to XML
  - Export Schema
  - Import from XML
- Versions
  - Create Version
  - View Versions
- Unlock File
- Copy XML
- Instance Generator
- Graph Representation
- Compare
- Rights

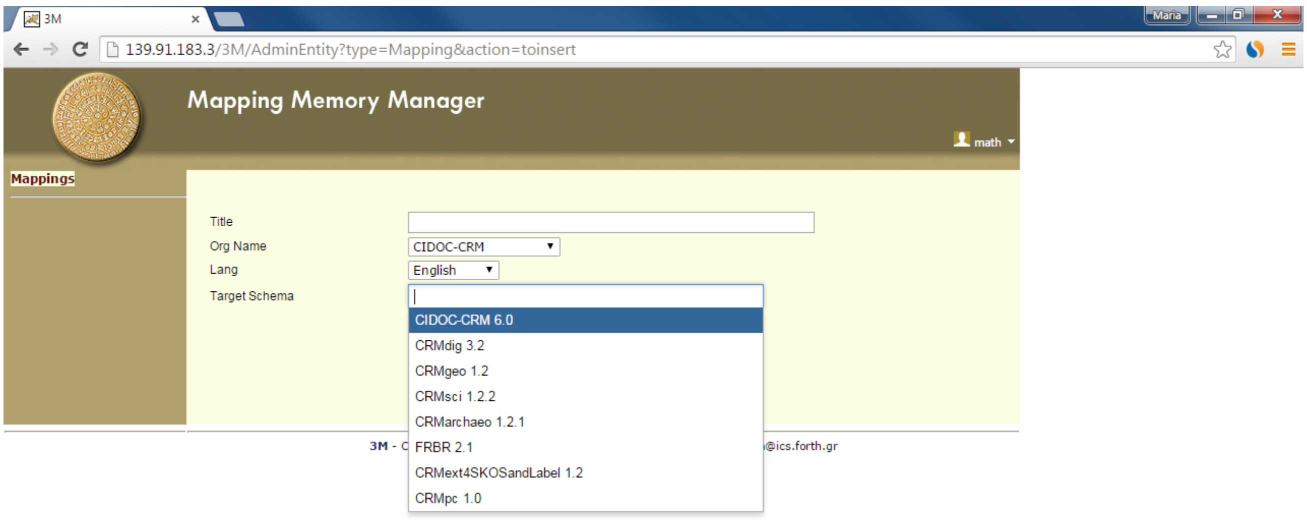
Title	Source Schema	General Description	Creator	Version Number	Status	Id
Auckland Museum Object	Object	To map the Auckland Museum's collection data to CRM!	Adamrmor	1	Unpublished	Mapping/182
RijksmuseumDocument			DOldman	1	Unpublished	Mapping/194
British Museum (DOldman)	RDFS		DOldman	1	Unpublished	Mapping/2
Auckland Museum Testing Version	Object	To map the Auckland Museum's collection data to CRM!	DOldman	1	Unpublished	Mapping/203
Rijksmuseum - 100 records version 2	Rijks100May15	The core Rijksmuseum object mapping	DOldman	1	Unpublished	Mapping/204
ResearchSpace	Rijksmuseum	A test mapping file for workshop	DOldman	1	Unpublished	Mapping/210

139.91.183.3/3M/ListEntity?type=Mapping&status=allunpublished#

1. Create New

When you create a new mapping you are asked to give it a title, select one or more of the proposed Target Schema and then click **Finish**. At least one schema must be selected (normally CIDOC CRM). 3M proposes a list of Target Schemata but it is not restricted to these. Later on, the user can load more Target Schemata (see Info Tab) or even delete selected ones.

If a new mapping is created successfully then the id of the new mapping is displayed. You can use the mappings link to return to the main screen.



3M

139.91.183.3/3M/AdminEntity?type=Mapping&action=toinsert

Mapping Memory Manager

math

Mappings

Title	
Org Name	CIDOC-CRM
Lang	English
Target Schema	

Target Schema options:
- CIDOC-CRM 6.0
- CRMdig 3.2
- CRMgeo 1.2
- CRMsci 1.2.2
- CRMarchaeo 1.2.1
- FRBR 2.1
- CRMext4SKOSandLabel 1.2
- CRMpc 1.0

3M - C … @ics.forth.gr

2. View

View provides other users with the opportunity of seeing mappings that they do not have editing rights to. The view is the same as the view mode in an editable mapping. Just select a mapping from the list so that it is highlighted and choose **View** from the main menu.

3. Edit

This options allows an authorised user (with edit rights) to edit the mapping and its associated settings. The required mapping must be selected in the mapping list.

4. Delete

Deletes a highlighted mapping permanently from the system. The action must be confirmed by clicking OK from the pop up window that appears, but can be cancelled. The creator or any editor can delete the mapping file from the system.

5. More
    a. Request for Publication

Request the publication of an "unpublished" mapping that the user has finished editing. In order to perform this action the user should select the desired "unpublished" mapping from the list and then choose from the upper part of the central region the action ***Request for Publish*** from the dropdown menu ***More***. A message window pops up asking for confirmation or not of the action. If he/she wishes to proceed to the publication of the mapping, he/she clicks the OK button and then the system shows a message informing the user that the mapping is now pending for publishing. 3M

If the user wishes to cancel the request for publish, he/she clicks the Cancel button and automatically the action is cancelled. The system administrator will decide if the mapping will be published. Published mappings cannot be edited anymore and they appear in the CIDOC-CRM site http://www.cidoc-crm.org/mapping_technology.html

b. Export to XML

Export a specific mapping at the local file system. In order to perform this action the user should select the desired mapping from the list and then click to the action ***Export to XML*** from the dropdown menu ***More***. After the completion of this action, a zip file with the Id of the mapping (e.g. Mapping82.zip) is downloaded locally. The zip file contains the X3ML definition of the mapping in XML format and all the accompanying source and target schemata and records. Currently this action is necessary when the user wants to get the X3ML definition in order to invoke the X3ML engine (offline tool) that will transform the source records to rdf (target records).

c. Import from XML

Import a mapping that has been locally edited in the editor. In order to perform this action the user should select from the local file system the zip file that contains the X3ML definition of the mapping and all the accompanying source and target schemata and records that he aims to import and then click to the action ***Import from XML*** from the menu ***More***.

d. Create Version

Create a new version for a specific mapping. In order to perform this action the user should select the desired mapping from the list and then click the action ***Create Version*** from the dropdown menu ***More***. After the completion of this action, a version of the current state of the specific mapping and also of the referenced files is created. Then the user may precede working on the specific mapping.

e. View Versions

View the versions for a specific mapping. In order to perform this action the user should select the desired mapping from the list and then click the action ***View Versions*** from the dropdown menu ***More***. The result of this action is a list with the created versions and the user can either ***View*** or ***Export to XML*** a specific version.

f. Unlock File

Unlock a specific mapping. Editing locks a mapping. Normally when 3MEditor window is closed, mapping is unlocked. However, sometimes something goes wrong and the lock remains valid for the whole session of the user. In order to perform this action, the user should select the desired mapping from the list and then click the action ***Unlock File*** from the dropdown menu ***More***. The selected mapping is unlocked and can be used by other users that have write rights on it.

g. Copy XML

Aim of this action is to offer the editor the possibility to create a copy of a mapping. In order to perform this action the editor should select the desired mapping from the list and then click to the action ***Copy XML*** from the dropdown menu ***More***.

h. Generators

Edit a specific mapping in order to add instance and label generation functions. In order to perform this action the user should select the desired mapping from the list and then choose from the upper part of the central region the action ***Generators*** from the dropdown menu ***More***.

i. Graph Representation

Show graph representation for a specific mapping. In order to perform this action the user should select the desired mapping from the list and then choose from the upper part of the central region the action ***Graph Representation*** from the dropdown menu ***More***.

j. Compare

Show a "diff-like" representation for 2 selected mappings. In order to perform this action the user should select the action ***Compare*** from the dropdown menu ***More.*** Then, two mappings have to be selected and once "Compare" button is clicked, selected mappings are presented side-by-side and textual differences are highlighted.

k. Rights

Change the rights of a specific mapping. In order to perform this action the user should select the desired mapping from the list and then choose from the upper part of the central region the action ***Rights*** from the dropdown menu ***More***. A list with all the users will appear and the user can select the users that he wants to give write rights. When he selects all the desired users he must click on ***Finish.***

6. **Search**
This function offers to the user the capability to locate mappings. The user can either type the words he/she likes to search about, restricting in that way the results to those that contain the specific words, or he/she may group the results according to the status of the card (published, pending etc.) which is available in the dropdown menu next to the Search. Search is currently case sensitive.