ICT MyMedia Project 2008-215006

Metadata Enrichment Quickstart

17 June 2009

Public Document

1 Contents

Project acronym: MyMedia

Project full title: *Dynamic Personalization of Multimedia*

Work Package:	**2**

Document title:	**Metadata Enrichment Quickstart**
Version:	**N/A**
Official delivery date:	**N/A**
Actual publication date:	N/A
Type of document:	Open Source Software and Documentation
Nature:	Public

Authors:	Chris Newell, BBC
	Tim Stevens, BT

Approved by:	N/A

2 Target Audience

The intended target audience of this document are consumers of the MyMedia project software framework.

See the MyMedia softare license, included at the root of the distribution, for terms of use.

3 Introduction

The Metadata Enrichment (ME) features allow software components called Enrichment modules to be added to the core frameworks in order to allow enhanced generation of descriptive metadata.

In the context of this document, the term *data* refers to items such as TV programme or movie information such as title, availability time, cost etc., whereas *metadata* refers to additional descriptive information such as viewer comments, cross-references to other programmes etc. It is assumed here that data items are available and correct, and that metadata may initially be present, however it is unlikely to be complete (and indeed, 'complete' is a subjective term, since metadata is being used to describe subjective opinion). The goal of Metadata Enrichment is to provide a means for the system to improve the existing metadata to some degree.

Enrichment modules should generally add metadata to the database rather than modify existing data. This approach means that the original catalogue data is preserved allowing different enhancement techniques to be compared.

4 How To Implement

In general, the Enrichment modules will integrate with other parts of the MyMedia core software, and familiarity with the core is required. Initially, modules must be developed in C-Sharp, although a project goal is to support development in other languages via a client/server interface.

4.1 Configuration

The MyMediaRecommenderQuickstart document contains information on configuring the database and other options using the MyMediaConfiguration options.

4.2 Creating an Enrichment module

The following namespaces should be referenced by Enrichment modules:

```
MyMediaProject.RecommenderSystem.Core;

MyMediaProject.RecommenderSystem.Core.Databases;

DatabaseInterface;
```

Enrichment modules should sub-class the `CatalogItemFactory` class. After creating an instance of `CatalogItemFactory` they should call the `Initialize()` method.

The `CatalogItemFactory.DataContext` field provides assess to the `CatalogItemStoreDataContext` instance which is used for all interaction with the catalog database. In this document this will be referred to as the `dataContext`.

Each Enrichment module should be given a unique module identifier. This is recorded together with the time and date of each entry in the catalog. allowing the source of each catalog entry to be identified. The Enrichment module should use the following method to create a `Module` and use it as a parameter in all enrichment operations:

```
Module module = dataContext.CreateModule(...
```

4.3 Process History API

The Process History API is provided to allow Enrichment modules to determine and record whether or not they have processed an item of content. When an Enrichment module runs it can use the API to obtain a list of all the items it has not yet processed. In the case where the Enrichment module is part of an ordered sequence of processes it is also possible to obtain a list of items it has not processed but which have been processed by another module.

Two methods are provided in the `MyMediaDatabaseInterface` class:

```
List<int> getProcessedBy(int processedBy, int notProcessedBy)

List<int> getProcessedBy(int processedBy, int notProcessedBy, DateTime since)
```

where:

`processedBy`	is the moduleId of the module which must have processed the items. If this is set to -1 the parameter is ignored.
`notProcessedBy`	is the moduleId of the module which has not processed the items. If this is set to -1 the parameter is ignored.
`since`	is the time and date after which these requirements should apply. Any process history records before this will be ignored.

The methods return a list of item identifiers. Having obtained a list of unprocessed items the Enrichment module can process each one in turn, signaling that it has done so using the following method:

```
markEnriched(int itemId, int moduleId)
```

4.4 Data Interfaces

Typically, an enrichment interface will examine existing data and metadata within the system, and enhance it in some way, before writing it back to the database.

An object representing a catalog item can be obtained from `dataContext` using the following code:

```
CatalogItem item = dataContext.CatalogItems.SingleOrDefault(i => i.ItemId ==
itemId);
```

where `itemId` is an identifier obtained using the `getProcessedBy` methods.

The properties of this object provide access to the existing data and metadata fields (e.g. title, description, and genre).

4.4.1 Adding a description

To add a new description or a synopsis to a catalog item use the following procedure:

```
ItemDescription itemDescription = dataContext.CreateDescription(...
itemDescription.Module = module;
item.AddDescription(dataContext, itemDescription);
```

4.4.2 Adding a keyword

To add a new keyword or tag to a catalog item use the following procedure:

```
Keyword keyword = dataContext.CreateKeyword(...
keyword.Module = module;
item.AddKeyword(dataContext, keyword, type, module);
```

4.4.3 Adding a link to related material

To add a new link to some material related to the item (e.g. a still image, web page, or programme transcript) use the following procedure. Each item of related material is identified by a URL. The nature of the relationship is signalled by a human-readable name and should be identified by a unique URI (e.g. using the TV-Anytime "HowRelated" classification scheme [1]).

```
Relationship relationship = dataContext.CreateRelationship(name, uri);
relationship.Module = module;
item.AddRelationship(dataContext, relationship, relatedItemUrl, module);
```

4.4.4 Adding a general item property

Any properties that are not specifically supported by the internal data model can be signalled using a generic property mechanism which specifies a property name / property value pair. Use the following procedure to add a Property to a catalog item:

```
Property property = dataContext.CreateProperty(propertyName, propertyValue);
property.Module = module;
item.AddProperty(dataContext, property);
```

4.4.5 Submitting changes

When the enrichment of an item has been completed the changes should be submitted by calling the following method:

```
dataContext.SubmitChanges();
```

The Enrichment module should then call the `markEnriched` method.

4.5 Notification Interfaces

Enrichment modules also require a means to be alerted when other components add new information.

5 References

[1] ETSI TS 102 822-3-1 (V1.4.1): "Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 3: Metadata; Sub-part 1: Phase 1 - Metadata schemas".