ICT MyMedia Project
2008-215006

Recommender Quickstart

17 June 2009

Public Document

1 Contents

Project acronym: MyMedia

Project full title: *Dynamic Personalization of Multimedia*

Work Package:	**2**

Document title:	**Recommender 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:	Rich Hanbidge, EMIC
Approved by:	N/A

2 Target Audience

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

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

3 Introduction

This document describes how to create a simple recommender system using the MyMediaProject software framework.

4 How To Implement

4.1 Requirements

- A basic understanding of recommender systems
- A working knowledge of managed code, specifically C-Sharp
- Microsoft Visual Studio 2008
    - Visual C# 2008 Express Edition (free)
    - http://www.microsoft.com/express/download/
- Microsoft SQL Server Express Edition(free)
    - http://www.microsoft.com/sqlserver/2005/en/us/express.aspx
    - Premium editions of Sql Server can be used as well
- .Net Framework, Version 3.5 or equivalent (free)
    - http://www.microsoft.com/downloads/details.aspx?FamilyID=333325fd-ae52-4e35-b531-508d977d32a6&DisplayLang=en
- The MyMediaProject Recommender Framework
- http://www.mymediaproject.org/Framework.aspxSample MovieLens data
    - http://www.grouplens.org/node/73
    - ***Be sure to view the license in the README files to ensure use of the MovieLens data is appropriate for your situation***

4.2 Scenario

This sample will create a simple movie recommender, using the popular “MovieLens” data set. This sample follows the “MovieLensRecommender” sample in the source code distribution.

4.3 Project creation

1. Start Visual Studio 2008

2. Create a new project called “MyMediaDemo”
    a. Click ‘File’ -> ‘New Project’
    b. Select ‘Console Application,’ or the project type of your choosing
    c. As name Enter ‘MyMediaDemo’
3. Add a reference to ‘MyMediaProject.RecommenderSystem.Core.dll’
    a. Go to the ‘Solution Explorer’
    b. Right click on the ‘MyMediaDemo’-solution (**not** the project) and select ‘Add Reference’
    c. Browse to the installation directory of the MyMediaProject Recommender Framework
    d. Browse to ‘MyMediaProject.RecommenderSystem.Core.dll’ and click the ‘OK’ button to reference this assembly.
4. Add a reference to ‘MyMediaProject.RecommenderSystem.Framework.dll’
    a. Right click on the ‘MyMediaDemo’-solution (**not** the project) and select ‘Add Reference’
    b. Browse to the installation directory of the MyMediaProject Recommender Framework
    c. Browse to ‘MyMediaProject.RecommenderSystem.Framework.dll’ and click the ‘OK’ button to reference this assembly.

4.4 Importing the data

4.4.1 Creating the catalog factory

The recommender framework has a notion of a catalog (item) factory. This factory has access to the internal catalog store. Once imported by the factory each item is automatically stored in databases for persistence. Note that the framework has no information about the source of the data. A concrete implementation of a catalog factory must inherit from ‘CatalogItemFactory’.

1. In the ‘Solution Explorer’ right click on the ‘Catalog’ folder and select ‘Add’ -> ‘Class’
2. Name the class ‘MovieLensDataFactory,” and have it inherit from MyMediaProject.RecommenderSystem.Core.CatalogItemFactory.
3. In the ‘Solution Explorer’ double click on ‘MovieLensDataFactory.cs’
    a. Create a new “Guid” for your CatalogItemFactory, along with a “string” describing the class.
        i. This guid and description must be passed to the CatalogItemFactory constructor when the class is created.
    b. Expose a method to import the data, called “ImportData” for this example.
        i. The helper methods to interact with the database are provide by the DataContext property exposed by the CatalogItemFactory base class your importer has inherited from.
            1. For every movie in the source data, use CatalogItemFactory::DataContext::CreateItem to produce a new CatalogItem class.
            2. Associate the title to the class using the CatalogItem::AddTitle() method.

        ii. Once all of the CatalogItems have been added to the database, call the base class method, CatalogItemFactory::SubmitChanges() to persist this data into the MyMediaFramework data storage solution.
    c. See "RecommenderFramework\MovieLensRecommender\Catalog\MovieLensDataFactory.cs" for a sample implementation.
4. For more detailed information, see the document, "MyMedia Framework How To CatalogImporter".

4.4.2 Importing Users

Users can be addd to the system in two ways; first, but calling methods directly on the Recommender class (see below), or in bulk, as will be discussed now.

In the "ImportData" method, defined above, add additional functionality to insert users from the data set.

1. For every user in the data set, create a new "User" class.
    a. Set properties, as appropriate on the User class
2. Queue the User for insertion into the framework, by calling CatalogItemFactory:: DataContext::Users::InsertOnSubmit
3. Once all of the users are queued for insertion, call CatalogItemFactory::DataContext::SubmitChanges to persist them to the storage solution.

4.4.3 Importing Rating Data

In the MyMediaFramework, all interactions between User and CatalogItems are considered "UserActions." UserAction is a base class for specific types of interactions. The "RatingAction" captures a User's explicit interest level in a CatalogItem. This is the appropriate type of UserAction to use for the MovieLens data set.

Also in the "ImportData" method, defined above, add additional functionality to insert ratings from the data set.

1. For ever rating in the data set, create a new RatingAction
    a. Set the CatalogItem (in this case, the item ID directly from the dataset is sufficient)
    b. Set the User (it is also appropriate here to simply use the User id from the dataset)
    c. Finally, set the RatingValue.
2. Queue the RatingAction for insertion by calling CatalogItemFactory::DataContext::UserActions::InsertOnSubmit
4. Once all of the ratings have been processed, call CatalogItemFactory::DataContext::SubmitChanges to persist them to the storage solution.
    a. Note; for large data sets, it may be necessary to submit subsets of the data, due to memory constraints on your system. A good guideline is to insert after every 10-50K entries.

4.4.4 Creating the recommender class

The recommender class is the single point of contact between the application frontend and the backend. It wraps the “real“recommender with its factory, flows and engines and only publishes the function subset that is needed by the application. This class should inherit from the Recommender class in the core framework.

1. In the ‘Solution Explorer’ right click on the ‘MyMediaDemo’-project and select ‘Add’ -> ‘Class’
2. Name the class ‘Recommender,’ and have it inherit from MyMediaProject.RecommenderSystem.Core.Recommender
3. In the ‘Solution Explorer’ double click on ‘Recommender.cs’
    a. The recommender class must define a unique Guid, which is used to define the default storage options, as well as a string, providing a short description of the recommender.

4.4.5 The default database parameters can be overridden by modifying the “DBConnectionString” property in the configuration. See section 4.7.1, SQL Server Installations

The framework is designed to work with the free Microsoft Sql Server “Express” 2005 edition. Please see the requirement section of this document for more details. It is important to note that Sql Express has a database size limit of 4GB. Larger scale solutions will require a premium version of Sql.

Different editions of the database server can be used by setting the appropriate Configuration information in the MyMedia.mmconf.xml file. Specifically the “DBConnectionString” property. See the Recommender Configuration discussion, in section 4.7.2.

Further, the framework will support any other Sql-Compliant database, by setting the appropriate connection information, per above.

        i. Recommender Configuration, for more information on this topic.
    b. See “\RecommenderFramework\MovieLensRecommender\Recommender.cs” for a sample implementation.

4.5 Getting Recommendations

At this point, we have created the data importer, and a basic recommender class to expose the core functionality. Now we will discuss how to generate recommendations for the user.

There are two basic components necessary for generating recommendations. These are “engines” and “flows.” The engine class is the implementation of the algorithm used to determine what is interesting to the user. The flow encapsulates the engine, along with other options, to abstract the recommendation process. The flow concept allows for multiple recommendation approaches to exist in the same application. This can be used, for example to support “A-B” testing to compare algorithms.

4.5.1 Flows

The recommender “Flow” has two critical components. The hybridizer and the engine. The hybridizer combines results from multiple engines, to produce recommendations. The engine or engines, represent the algorithm used to generate recommendations.

For this simple example, we will use a single engine hybridizer, present in the framework. For the engine, we will also use a framework provided “Matrix Factorization” engine. In the following steps we will create a recommender flow, using a matrix factorization engine, and a single-engine hybridizer.

1. In the ‘Solution Explorer’ double click on ‘Recommender.cs’
2. In the recommender class, add a class member variable of type “Flow.”
3. In the recommender class, override the method called “Initialize”
    a. Start with calling the base implementation of Initialize, using “base.Initialize()”. This will perform necessary initialization of the framework.
    b. Next, initialize the flow class member variable with a new Flow object
    c. In the flow object, add a new MatrixFactorizationEngine to the flow, by using the Flow.AddEngine method
    d. Now add a SingleEngineHybridizer, by setting the Hybridizer property on the flow
    e. Finally, register the flow with the recommender, using the “base.RegisterFlow” method
4. See “\RecommenderFramework\MovieLensRecommender\Recommender.cs” for a sample implementation.

You have now added a recommender flow to your recommender class, which can be used to generate recommendations.

4.5.2 Engines

For more information on engines, see the “MyMedia Framework How To Engine” document.

4.5.3 Generating Recommendations

At this point, you have created a recommender Flow, which encapsulates a recommendation approach, with an Engin and Hybridizer. To get recommendations from the system, you simply call the Recommender.Predict method, specifying the “FlowId” of the recommender flow you would like to use. This will call the engine(s) registered in the flow to create a set of “Prediction” objects, which contain the relevant information on the recommendations.

Care should be taken to restrict the catalog to only the relevant items, for performance reasons. For example, this could restrict the movies to only those in a particular Genre.

4.5.3.1 For more information on flows, please see the detailed Database Configuration

The database connection string used by the framework is configurable via the “DBConnectionString” property in the MyMedia.mmconf.xml file. The database must be of the schema used by MyMedia.

The default behavior is to connecto to an instance of “SqlExpress” running on the local machine. The database name is produced by merging the program name of the Recommender instance, with the program id. For example, “MovieLensRecommender.a53d4502-5da1-4dfe-81ff-cadf03588064”. By default, this database will be created if it does not exist.

Flows discussion, section 4.7.2.1, in Advanced Scenarios.

4.5.4 Providing Feedback

New ratings or user behavior are added to the framework via the Recommender.ProcessFeedback method. Simply create a new “UserAction” object, of the appropriate type. For explicit ratings, use the “UserRating” Class.

For more information on flows, please see the detailed UserActions and Data discussion, section 4.7.6, in Advanced Scenarios.

4.6 Creating a UI

The specifics of creating a recommender UI are beyond the scope of this document. However, a very simple UI should provide a list of users, their rated items, and the predictions for those users. See the MovieLensRecommender project for an “end-to-end” example.

4.7 Advanced Scenarios

4.7.1 SQL Server Installations

The framework is designed to work with the free Microsoft Sql Server “Express” 2005 edition. Please see the requirement section of this document for more details. It is important to note that Sql Express has a database size limit of 4GB. Larger scale solutions will require a premium version of Sql.

Different editions of the database server can be used by setting the appropriate Configuration information in the MyMedia.mmconf.xml file. Specifically the “DBConnectionString” property. See the Recommender Configuration discussion, in section 4.7.2.

Further, the framework will support any other Sql-Compliant database, by setting the appropriate connection information, per above.

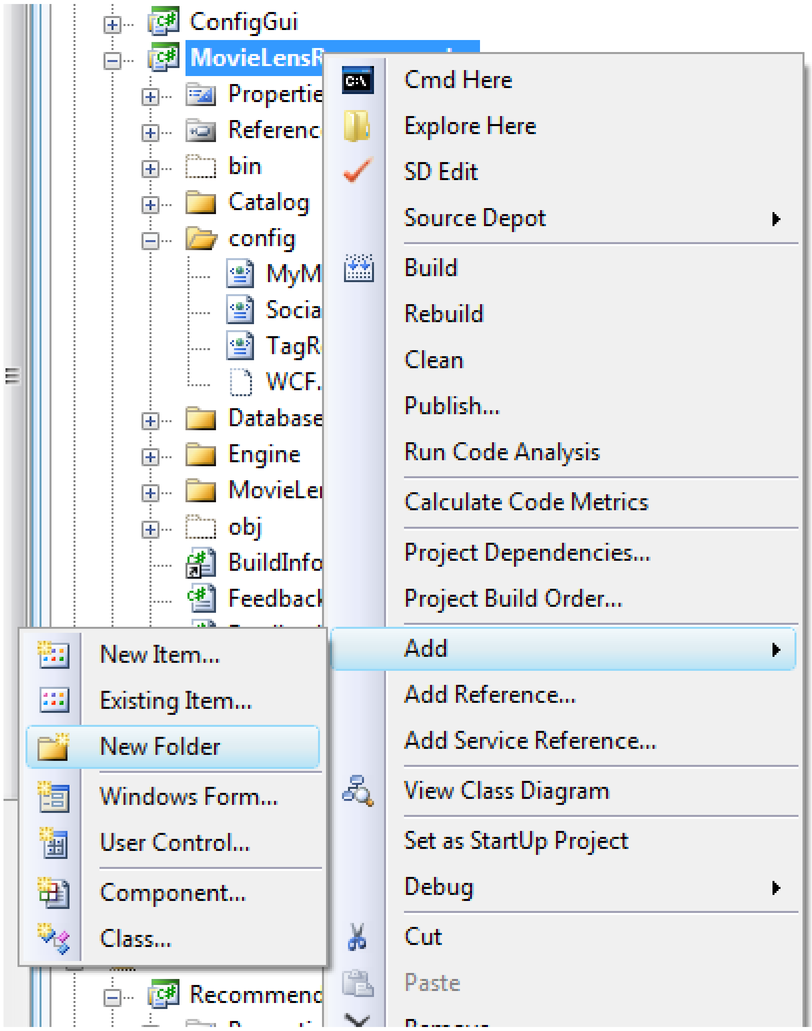
4.7.2 Recommender Configuration

The framework provides a configuration file for setting various database, proxy, and other settings. See the “config” directory in any of the projects of the framework for the various configuration files. These are of the format <FileName>.mmconf.xml.

To set application specific settings, a copy of the mmconf.xml file should be created in your application project in a folder called “config.” This file can then be modified for your application, and copied to the execution directory. See below for specific steps.

1. Create a project folder called “config” in your application project

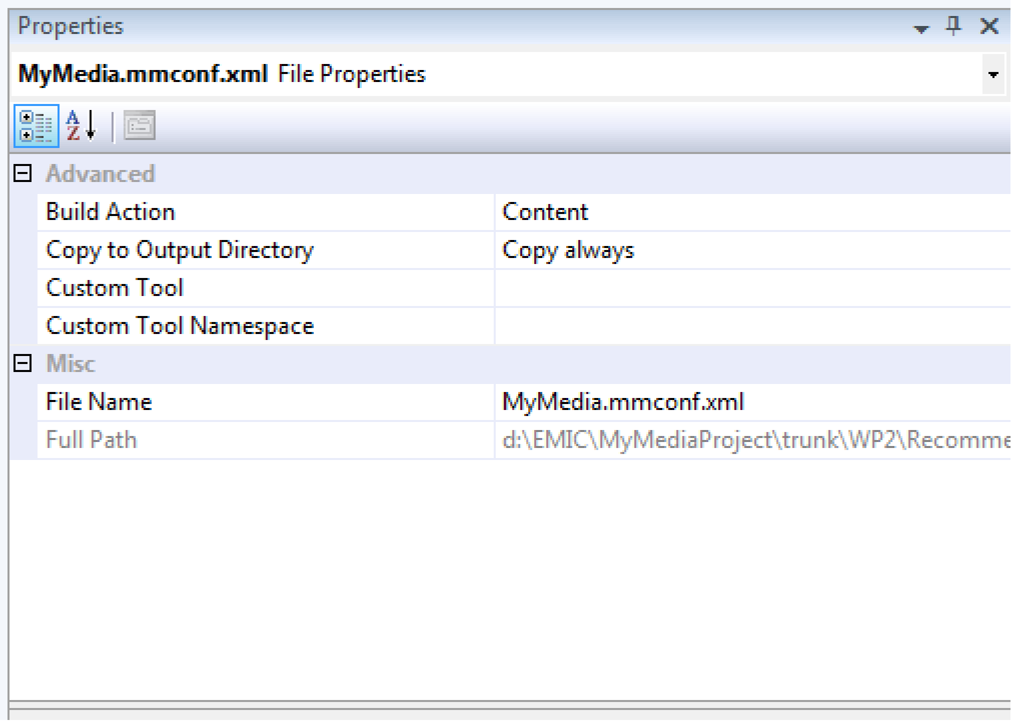
a. Right-Click on the project
    i. Select Add->New Folder



    ii. Name the folder “config”

b. Copy the appropriate mmconf.xml file(s) to this directory, and include them in your project
    i. Right click on the “config” folder
    ii. Select Add->Existing Item
        1. This will present you with a file explorer dialog, in which you can navigate to the mmconf.xml files

c. Set each of the files with the “content” Build Action, and the Copy Always setting.



Properties

MyMedia.mmconf.xml File Properties

Advanced	
Build Action	Content
Copy to Output Directory	Copy always
Custom Tool	
Custom Tool Namespace	
Misc	
File Name	MyMedia.mmconf.xml
Full Path	d:\EMIC\MyMediaProject\trunk\WP2\Recomme

d. Now, set each of the properties in the XML file to the appropriate values

    i. **Note – removing properties from this file is not allowed, and may cause undesired results.**

4.7.2.1 Database Configuration

The database connection string used by the framework is configurable via the “DBConnectionString” property in the MyMedia.mmconf.xml file. The database must be of the schema used by MyMedia.

The default behavior is to connecto to an instance of “SqlExpress” running on the local machine. The database name is produced by merging the program name of the Recommender instance, with the program id. For example, “MovieLensRecommender.a53d4502-5da1-4dfe-81ff-cadf03588064”. By default, this database will be created if it does not exist.

4.7.3 Flows

The recommender Flow encapsulates a recommendation strategy, and the framework supports many Flows in the same application at the same time. This can be a powerful tool for “A-B” testing where users are randomly assigned different recommendation algorithms as an advanced algorithm assessment procedure. This can also be used to apply different recommendation algorithms at different times.

In addition to the previously discussed features, the Flow also exposes two other important concepts. A “PreFilter” and “PostFilter.”

The PreFilter is a fixed CatalogItem query restriction, executed before the query is passed to the recommendation Engines. This acts as a filter for items which are available for recommendation in a given flow. For example, this can be used to restrict a television recommender to the shows which are available in the next 2 hours.

The PostFilter is a fixed result filter, which limits the Prediction objects returned by the Recommender.Predict method on that particular Flow instance. This can be used to limit the Predictions to those the user will really like, but only returning high values.

To add either of these filters to the Flow, simply call the appropriate Add* method, passing in a class which implements the IPreFilter or IPostFilter interface.

4.7.4 Hybridization

Hybridization is an approach to leverage the strength of various recommendation engines. This allows results from multiple engines to be combined to produce a better result. The framework allows new and plugable hybridization methods to be implemented to combine engine results.

To create a new Hybridizer, simply create a call implementing the IHybridizer interface. This can then be applied to the Flow.Hybridizer property on the Flow object.

4.7.5 Users

The User class provides a number of Properties relevant to applications and field trails. The automatically generated documentation in the software descripes these in detail.

The User class also exposes the notion of “UserProperty” to extend this further. This allows for arbitrary <name,value> pairs to be associated with a given User object. This could be extenstions to the framework such as phone numbers or other information.

To use this, simply define a well-known name string, such as “UserPhoneNumber,” which represents the property key. The value can then be any string, and will be persisted in the framework database.

4.7.6 UserActions and Data

A UserAction is the base object representing a User interaction with the system. There are a number of derived classes, such as RatingAction, which represent specific types of actions. The software documentation describes these in full detail.

The framework also provides an extensibility mechanism of <name,value> pairs to specifiy new type of UserActions. This is the OtherAction class. To create a new type of action, simply create an OtherAction class, with a well defined name, which is used as the key. The actionProperty can be any string, representing detailed information about that interaction.

5 FAQs

5.1.1 How to show the Solution Window?

Click on ‘View’ -> ‘Solution Explorer’

5.1.2 What is the difference between a project and a solution?

A solution can contain multiple projects. In Visual Studio a project must be a member of a solution or Visual Studio will create a default solution with the same name. In the ‘Solution Explorer’ the solution item is always the root Element. All projects that are member of the solution are listed at the hierarchy level of the tree.

5.1.3 How to switch to an use Quick Edit Mode in a command console window

1. Right click on the header of the window and choose ‘Properties’
2. Navigate to the ‘Options’ tab and check Quick Edit Mode

To copy in the Quick Edit mode select the text you want to copy and hit Enter/Return. Whenever you want to paste the string again click the right mouse button. The text will be pasted to the current text cursor location.

As the standard buffer of the console output is only 300 line, it might be necessary to increase it:

1. Right click on the header of the window and choose ‘Properties’
2. Navigate to the ‘Layout’ tab and set the screen buffer size height to a bigger value (e.g. 900)