

Mach-II Contact Manager DB

3/2/2006

Matt Woodward

mpwoodward@gmail.com

Skype: cfweekly

The Mach-II Contact Manager sample application is designed to provide those interested in learning Mach-II a simple sample application that uses a database, both for the purposes of seeing how a typical Mach-II application interacts with a database, but also to introduce some of the typical database-related objects that appear in OO applications.

Requirements

- ColdFusion MX 6.1 or higher
- Mach-II 1.1.0 (previous versions *will not work*)
- SQL Server 2005 (Express will work) or MySQL 5. The application will very likely work on earlier versions of these databases since it's such a simple, one-table database but these are the two databases on which I tested the application, and the databases for which create scripts are provided.

Installation

1. If you don't already have Mach-II 1.1.0 installed, make sure it's installed either in your web root in a directory called MachII or that you have created a mapping called MachII that points to the Mach-II framework files. See <http://www.mach-ii.com> for Mach-II installation assistance.
2. Unzip the contents of the zip file into the root of your web server or the ColdFusion built-in web server, which should create a directory in your web root called ContactManagerDB. By default the application will be expecting to reside in this directory. You can change this but you would also have to change the pathing for the CFCs in the XML configuration file, so unless there's a reason you can't use this directory name, keeping it as is will make installation simple.
3. Create the database in SQL Server or MySQL (again, I tested on SQL Server Express 2005 and MySQL 5) using one of the scripts provided in the database directory of the application. For SQL Server first create the database then run the SQL Server script to create the table needed. For MySQL the script provided is a MySQL dump and includes the create database command, so you can use the MySQL administrator and do a restore or do an import from the command line. See the respective documentation for these databases if you need help with this process. If you prefer to use another database creating it should be easy by looking at either of these two SQL scripts.
4. Create a datasource in the ColdFusion administrator that points to your database. By default the application will be looking for a datasource called M2ContactManager. You can call your datasource whatever you want, just make sure if you don't use the default that you update the dsn property in the mach-ii.xml file to match your datasource name. (More on that below.)
5. Load [http://YOUR_MACHINE\[:PORT\]/ContactManagerDB](http://YOUR_MACHINE[:PORT]/ContactManagerDB) in your browser and the application should come up. If it doesn't, review steps 1-4 and make sure everything is configured correctly.

Configuration

If you retain the default directory and datasource settings in the application itself everything will just run (at least that's the theory!). If you find yourself having to change some of these settings, please note the following:

- The datasource name may be changed on line 14 of config/mach-ii.xml. Simply change the value attribute to the name of your datasource as defined in the ColdFusion administrator and save the file.
- If you have to put the application somewhere other than ContactManagerDB in your web

root, you will have to update all instances of `ContactManagerDB.model.[CFC_NAME]` to reflect the correct directory path. You will also have to update line 31 of `filters/ContactBeanerFilter.cfc`. If you don't have to change the directory it's best to leave it as is.

Application Views

This application, like all Mach-II applications, is based on the Model-View-Controller (MVC) design pattern which separates the model (your CFCs) and controller (essentially your `mach-ii.xml` file and your listeners) from the view layer (the `.cfm` pages with which the user interacts). In most cases the view files will be quite simple because the data required by the views will be retrieved before the view file is involved in the request, meaning the amount of logic code on your view page will be limited to only presentation layer logic, such as looping over a query to display data to the user.

Mach-II also has the concept of multiple view files making up a single final page. In this application you will see that the pages use `mainTemplate.cfm` as the “wrapper” view around each individual page, which is placed in the event object as a `contentArg`. For example, if you examine the `showHome` event handler, the first view called is `mainMenu`, which is given a `contentArg` name of “content.” This is followed by the `mainTemplate` view which in addition to the repeated portions of the view code such as the navigation, simply outputs this `contentArg` by retrieving it from the event object by calling `event.getArg(“content”)`. The use of multiple views in this way can increase code reuse in the view layer and allows for a great deal of flexibility in your views as well.

Application CFCs

The purpose of this application is not only to provide a simple sample application that illustrates the basics of how Mach-II works, but also to introduce some of the classic data object patterns that are used in OO applications. The “triumvirate” of persisting and loading bean-type objects typically includes these objects (all can be found in the model directory):

- `Contact.cfc`: The Contact bean, which acts as a software model of the real-world contact data. This encapsulates the contact data into a single, easy-to-use object, and is also the object that will be populated from and passed to the Data Access Object (DAO).
- `ContactGateway.cfc`: The gateway object can be thought of as an interface to your database for queries that return multiple records. Methods such as `getAllContacts()` and `getRecentContacts()` will reside here, and very often methods in the gateway object will return a ColdFusion query object.
- `ContactDAO.cfc`: The Data Access Object (DAO) is again an interface to your database, but the DAO deals with single records as opposed to multiple records. The classic DAO pattern is to have this object contain four methods: `create()`, `read()`, `update()`, and `delete()`. These methods are described in greater detail in “The CRUD” below.

In addition to these CFCs, the `ContactListener` object plays an important role in the application. In many of the event-handler tags you will notice that the `ContactListener` is notified that it needs to call a method and return an argument. For example, in the `showHome` event, the `ContactListener` is notified to call its method `getRecentContacts()` and return that as a `resultArg` named `recentContacts`. What this accomplishes is that the `recentContacts` query object is placed in the event object, thereby making it available for use on the view for display purposes. In other cases the listener is notified but the method called doesn't return anything, which typically means that based on what happens inside the listener's method it announces the next event itself as appropriate.

The CRUD

Data Access Objects (DAOs) deal with single database records through Create, Read, Update, and Delete methods, hence the “CRUD” abbreviation for database interaction concerning single records. Each of these methods will typically take in a bean of the object it manages as an

argument and then perform the appropriate operation in the database.

- The create() method takes in a Contact bean as an argument that is populated with data and inserts the bean data into the database.
- The read() method takes in a Contact bean as an argument that is populated with an ID only, and the DAO then retrieves the details for the object from the database using this ID and sets the database values to the appropriate attributes in the object.
- The update() method takes in a Contact bean as an argument that is populated with data and updates the record in the database using the Contact bean's ID.
- The delete() method takes in a Contact bean as an argument that is populated with an ID only, and the DAO then deletes the record from the database using this ID.